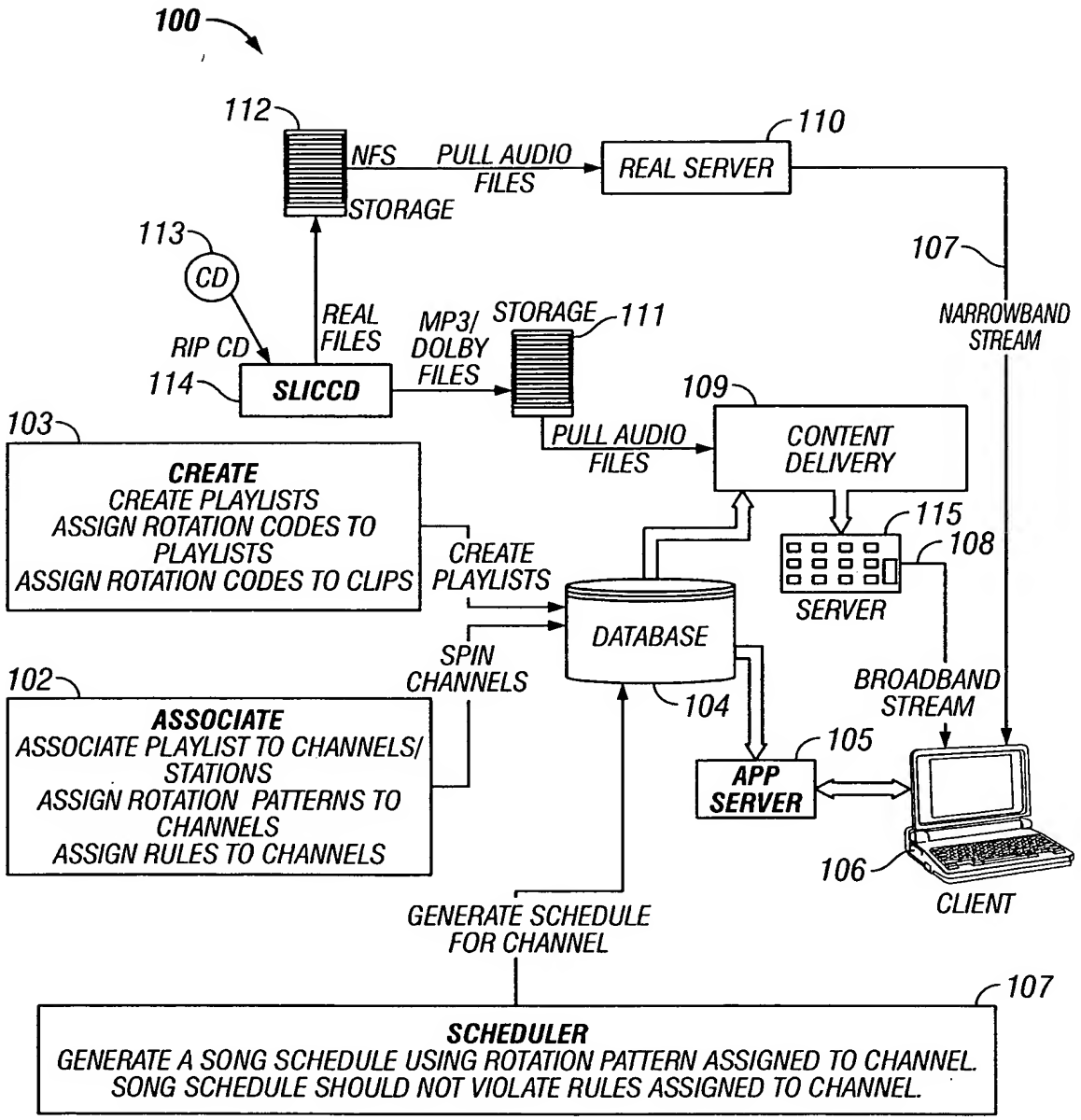


FIG. 1

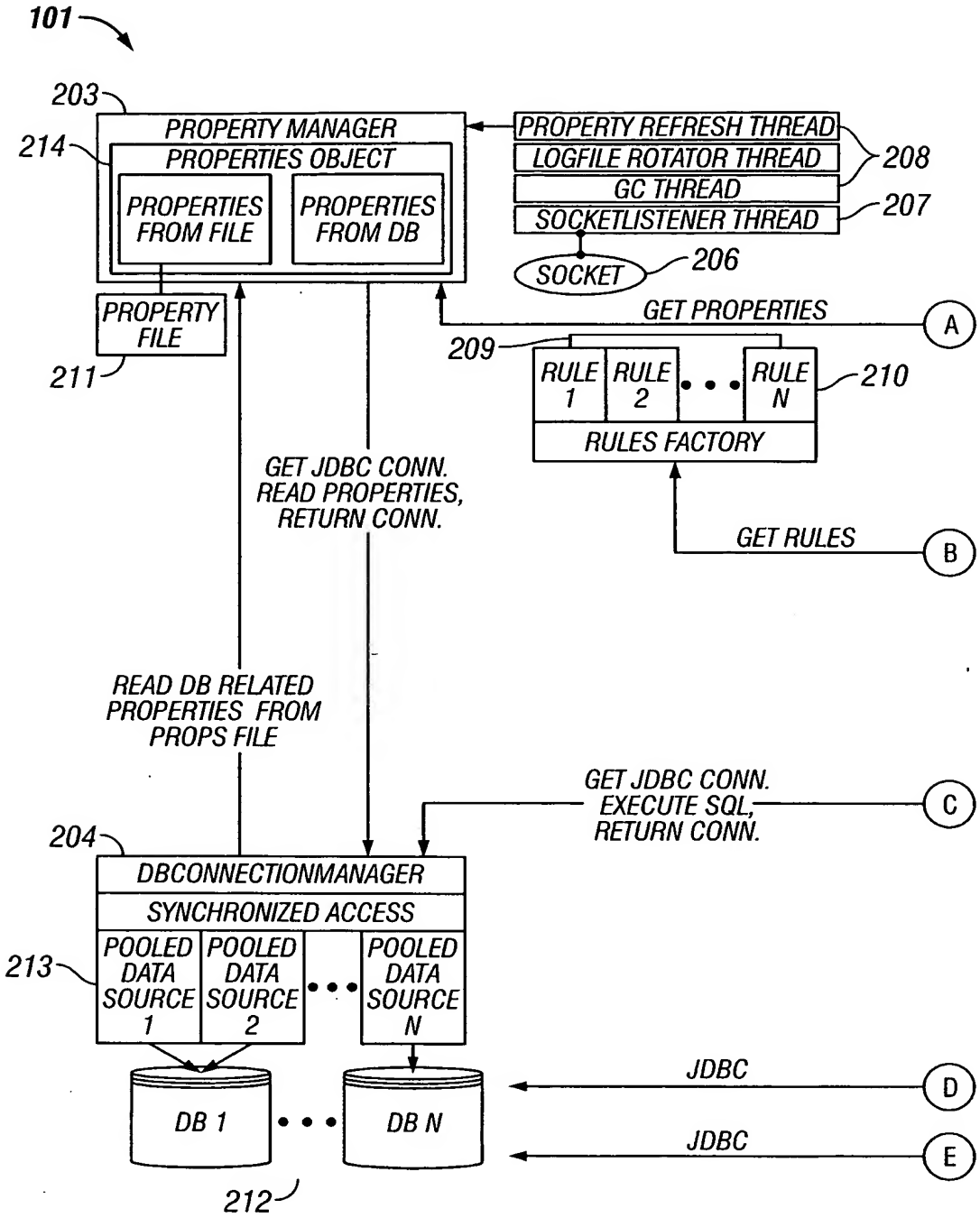

101
203
PROPERTY MANAGER
214
PROPERTIES OBJECT
PROPERTIES FROM FILE
PROPERTIES FROM DB
PROPERTY FILE
211
PROPERTY REFRESH THREAD
LOGFILE ROTATOR THREAD
GC THREAD
SOCKETLISTENER THREAD
208
207
SOCKET
206
GET PROPERTIES
A
209
RULE 1
RULE 2
RULE N
210
RULES FACTORY
GET RULES
B
GET JDBC CONN.
READ PROPERTIES,
RETURN CONN.
READ DB RELATED
PROPERTIES FROM
PROPS FILE
GET JDBC CONN.
EXECUTE SQL,
RETURN CONN.
C
204
DBCONNECTIONMANAGER
SYNCHRONIZED ACCESS
213
POOLED DATA SOURCE 1
POOLED DATA SOURCE 2
POOLED DATA SOURCE N
JDBC
D
DB 1
DB N
JDBC
E
212

FIG. 2

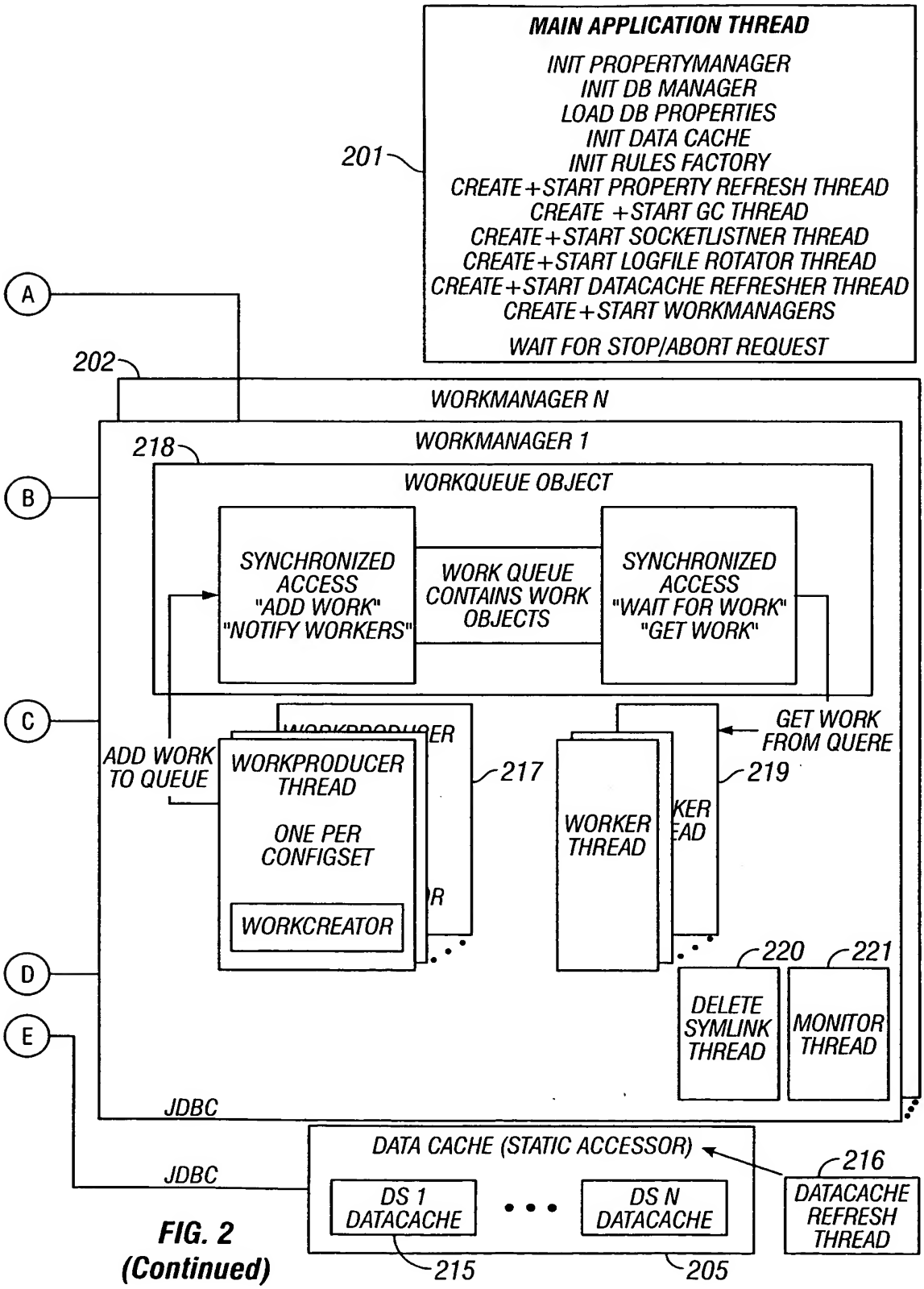


FIG. 2 (Continued)

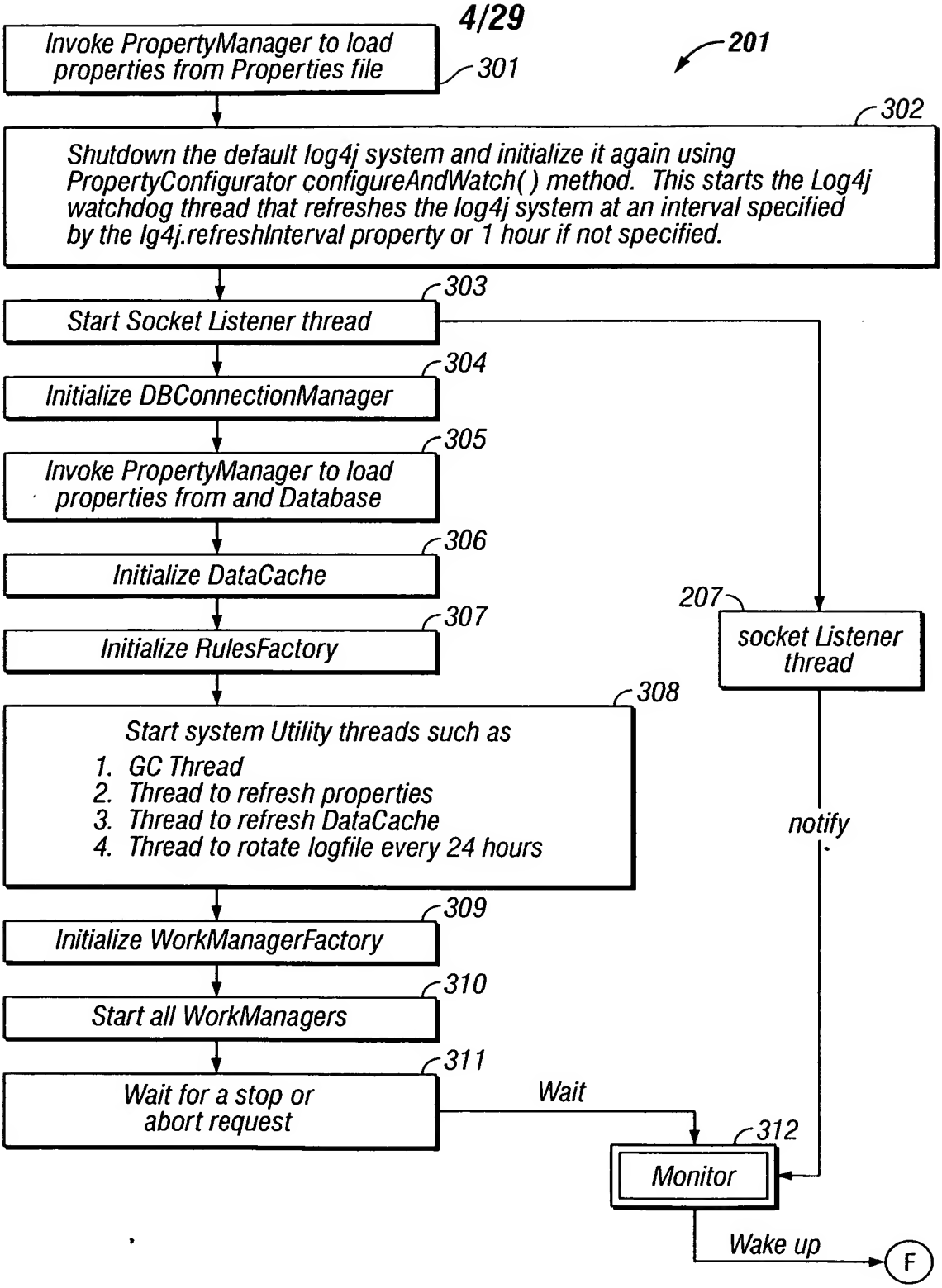

201
Invoke PropertyManager to load properties from Properties file
301
302
Shutdown the default log4j system and initialize it again using PropertyConfigurator configureAndWatch( ) method. This starts the Log4j watchdog thread that refreshes the log4j system at an interval specified by the lg4j.refreshInterval property or 1 hour if not specified.
303
Start Socket Listener thread
304
Initialize DBConnectionManager
305
Invoke PropertyManager to load properties from and Database
306
Initialize DataCache
307
Initialize RulesFactory
308
Start system Utility threads such as
1. GC Thread
2. Thread to refresh properties
3. Thread to refresh DataCache
4. Thread to rotate logfile every 24 hours
309
Initialize WorkManagerFactory
310
Start all WorkManagers
311
Wait for a stop or abort request
Wait
207
socket Listener thread
notify
312
Monitor
Wake up
F

FIG. 3

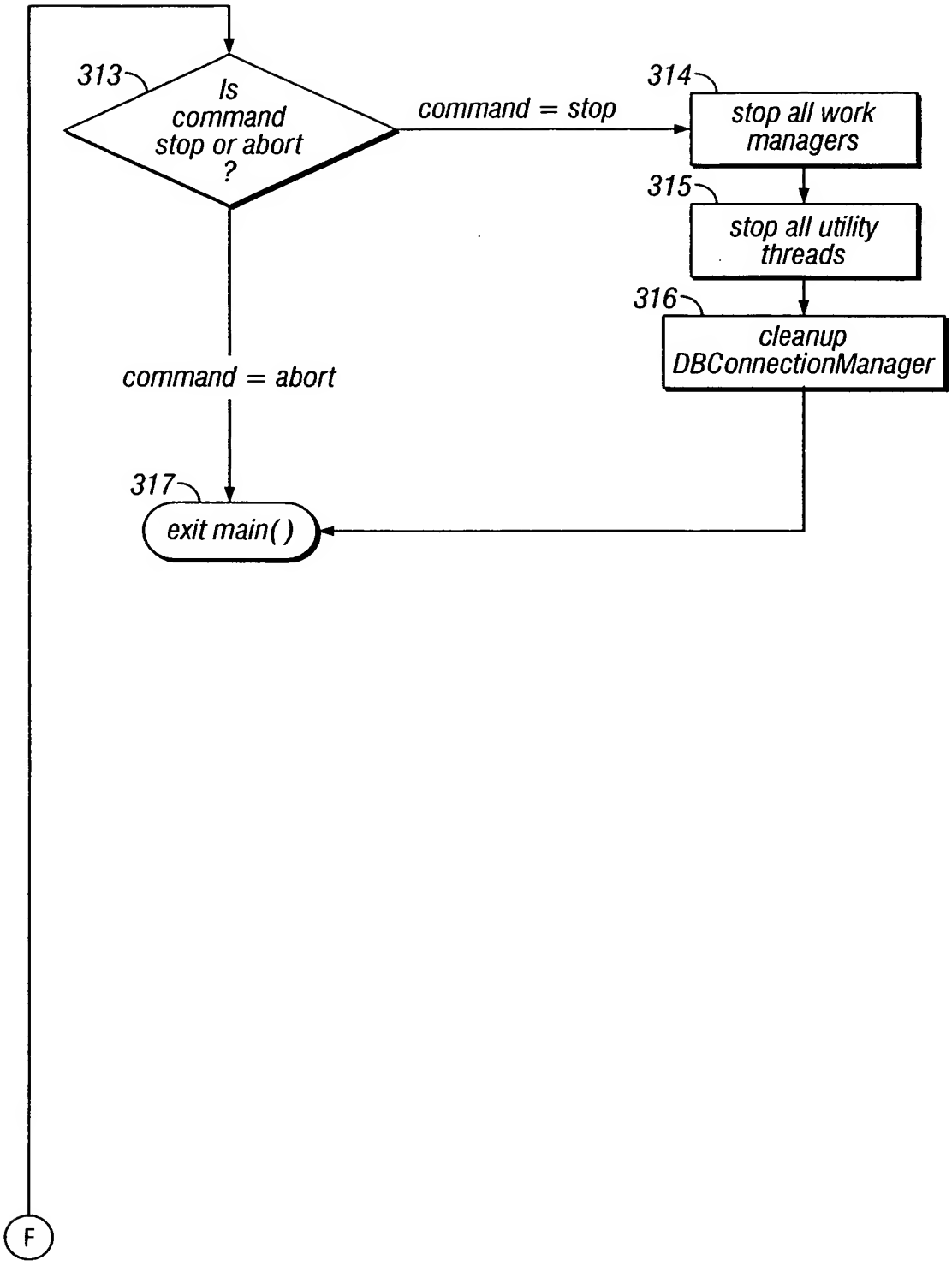


FIG. 3
(Continued)

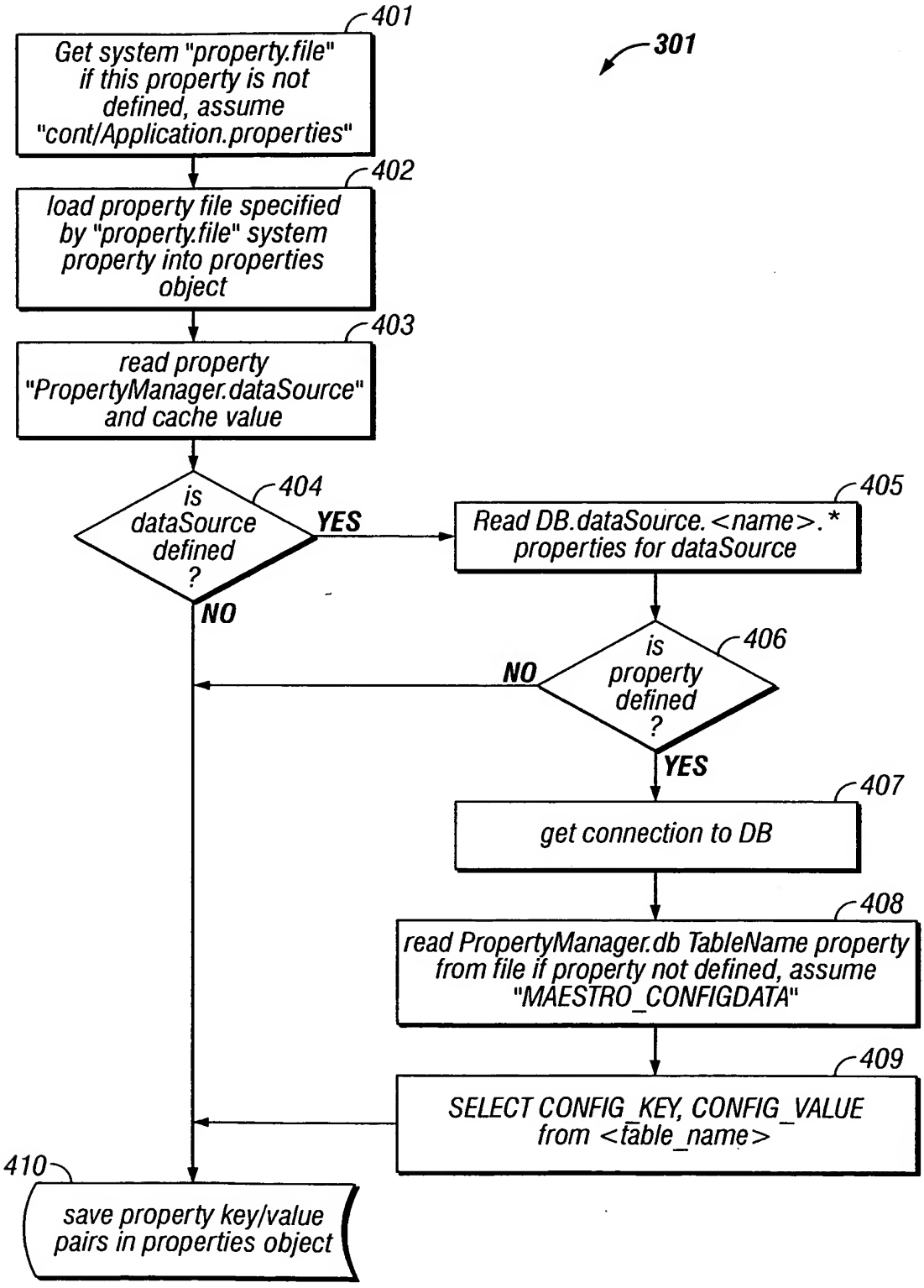

401
Get system "property.file" if this property is not defined, assume "cont/Application.properties"
301
402
load property file specified by "property.file" system property into properties object
403
read property "PropertyManager.dataSource" and cache value
404
is dataSource defined ?
YES
NO
405
Read DB.dataSource.<name>.* properties for dataSource
406
is property defined ?
NO
YES
407
get connection to DB
408
read PropertyManager.db TableName property from file if property not defined, assume "MAESTRO_CONFIGDATA"
409
SELECT CONFIG_KEY, CONFIG_VALUE from <table_name>
410
save property key/value pairs in properties object

FIG. 4

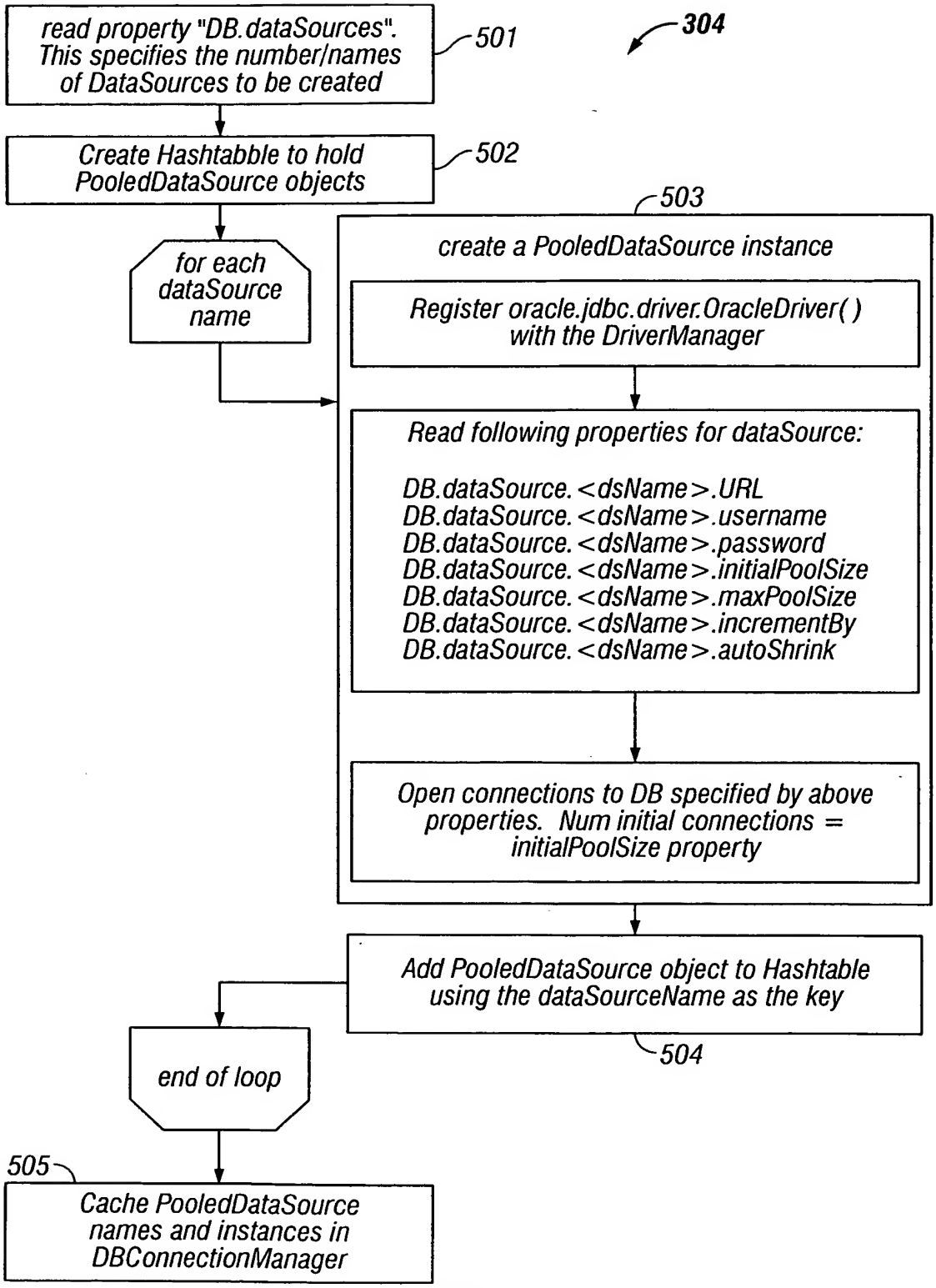

read property "DB.dataSources". This specifies the number/names of DataSources to be created
501
304
Create Hashtabble to hold PooledDataSource objects
502
503
for each dataSource name
create a PooledDataSource instance
Register oracle.jdbc.driver.OracleDriver( ) with the DriverManager
Read following properties for dataSource:
DB.dataSource.<dsName>.URL
DB.dataSource.<dsName>.username
DB.dataSource.<dsName>.password
DB.dataSource.<dsName>.initialPoolSize
DB.dataSource.<dsName>.maxPoolSize
DB.dataSource.<dsName>.incrementBy
DB.dataSource.<dsName>.autoShrink
Open connections to DB specified by above properties. Num initial connections = initialPoolSize property
Add PooledDataSource object to Hashtable using the dataSourceName as the key
504
end of loop
505
Cache PooledDataSource names and instances in DBConnectionManager

FIG. 5

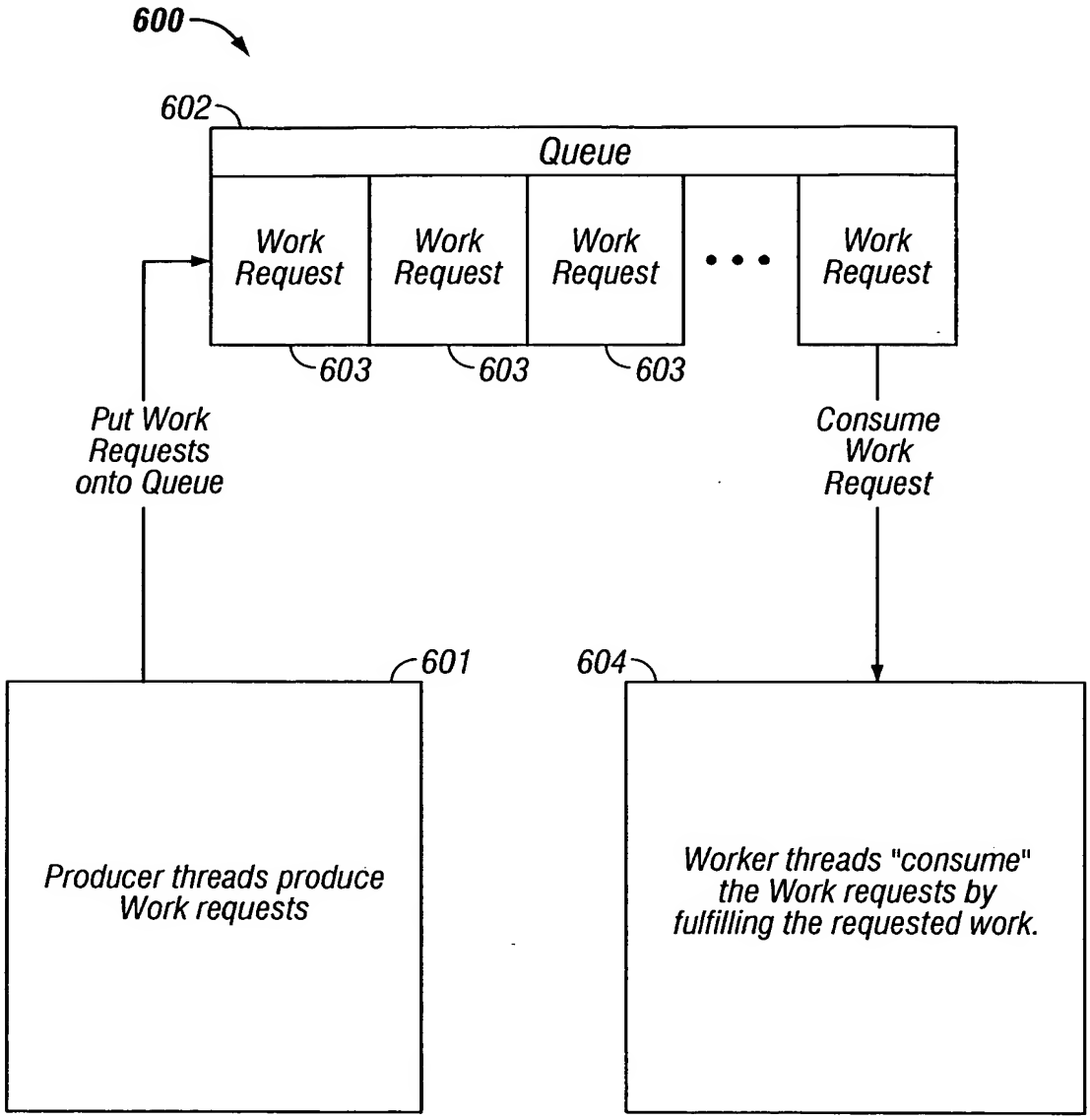

600
602
Queue
Work Request
Work Request
Work Request
Work Request
603
603
603
Put Work Requests onto Queue
Consume Work Request
601
604
Producer threads produce Work requests
Worker threads "consume" the Work requests by fulfilling the requested work.

FIG. 6

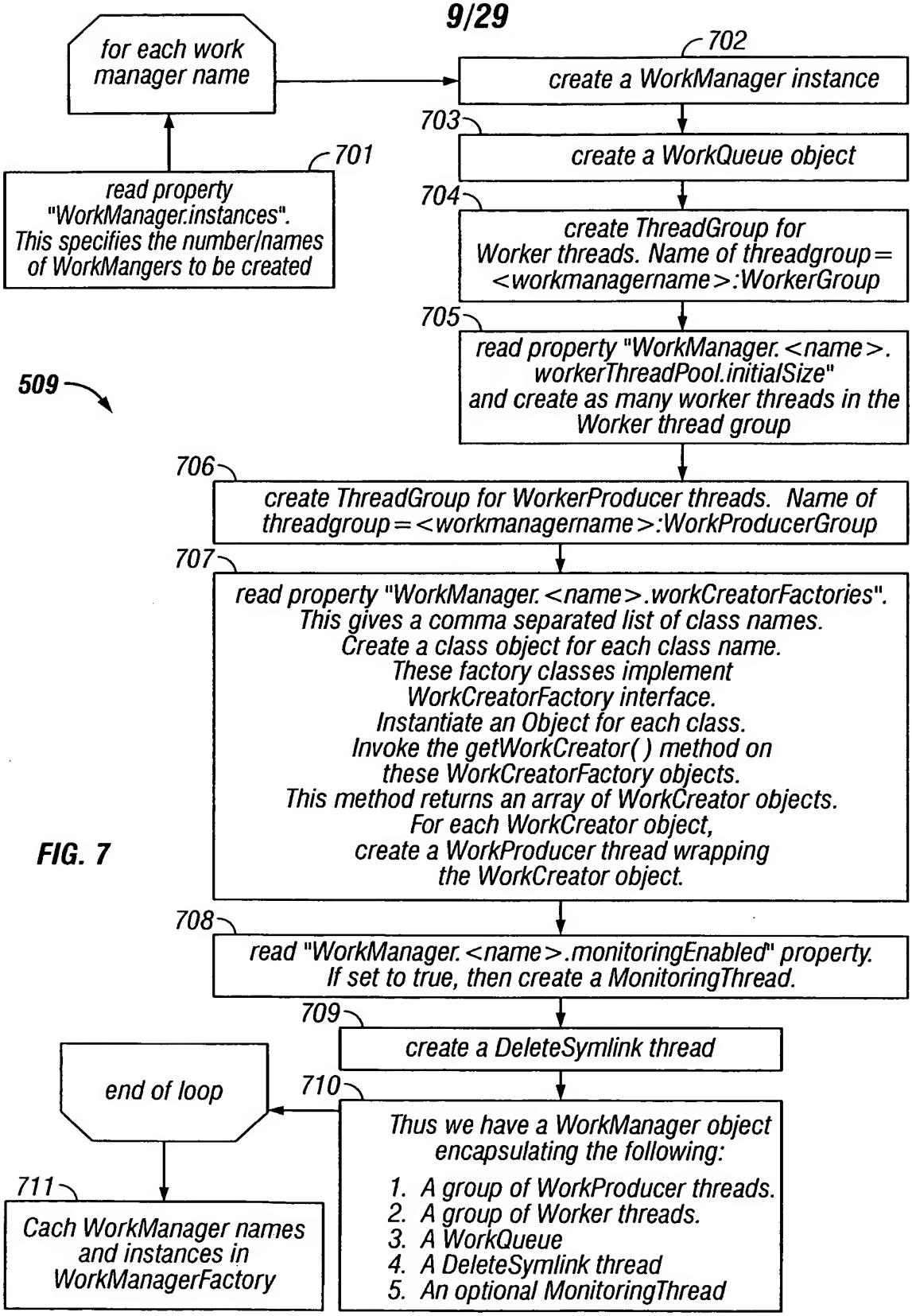

for each work manager name
701
read property "WorkManager.instances". This specifies the number/names of WorkMangers to be created
509
702
create a WorkManager instance
703
create a WorkQueue object
704
create ThreadGroup for Worker threads. Name of threadgroup = <workmanagername>:WorkerGroup
705
read property "WorkManager.<name>. workerThreadPool.initialSize" and create as many worker threads in the Worker thread group
706
create ThreadGroup for WorkerProducer threads. Name of threadgroup=<workmanagername>:WorkProducerGroup
707
read property "WorkManager.<name>.workCreatorFactories". This gives a comma separated list of class names. Create a class object for each class name. These factory classes implement WorkCreatorFactory interface. Instantiate an Object for each class. Invoke the getWorkCreator( ) method on these WorkCreatorFactory objects. This method returns an array of WorkCreator objects. For each WorkCreator object, create a WorkProducer thread wrapping the WorkCreator object.
708
read "WorkManager.<name>.monitoringEnabled" property. If set to true, then create a MonitoringThread.
709
create a DeleteSymlink thread
710
Thus we have a WorkManager object encapsulating the following:
1. A group of WorkProducer threads.
2. A group of Worker threads.
3. A WorkQueue
4. A DeleteSymlink thread
5. An optional MonitoringThread
end of loop
711
Cach WorkManager names and instances in WorkManagerFactory

FIG. 7

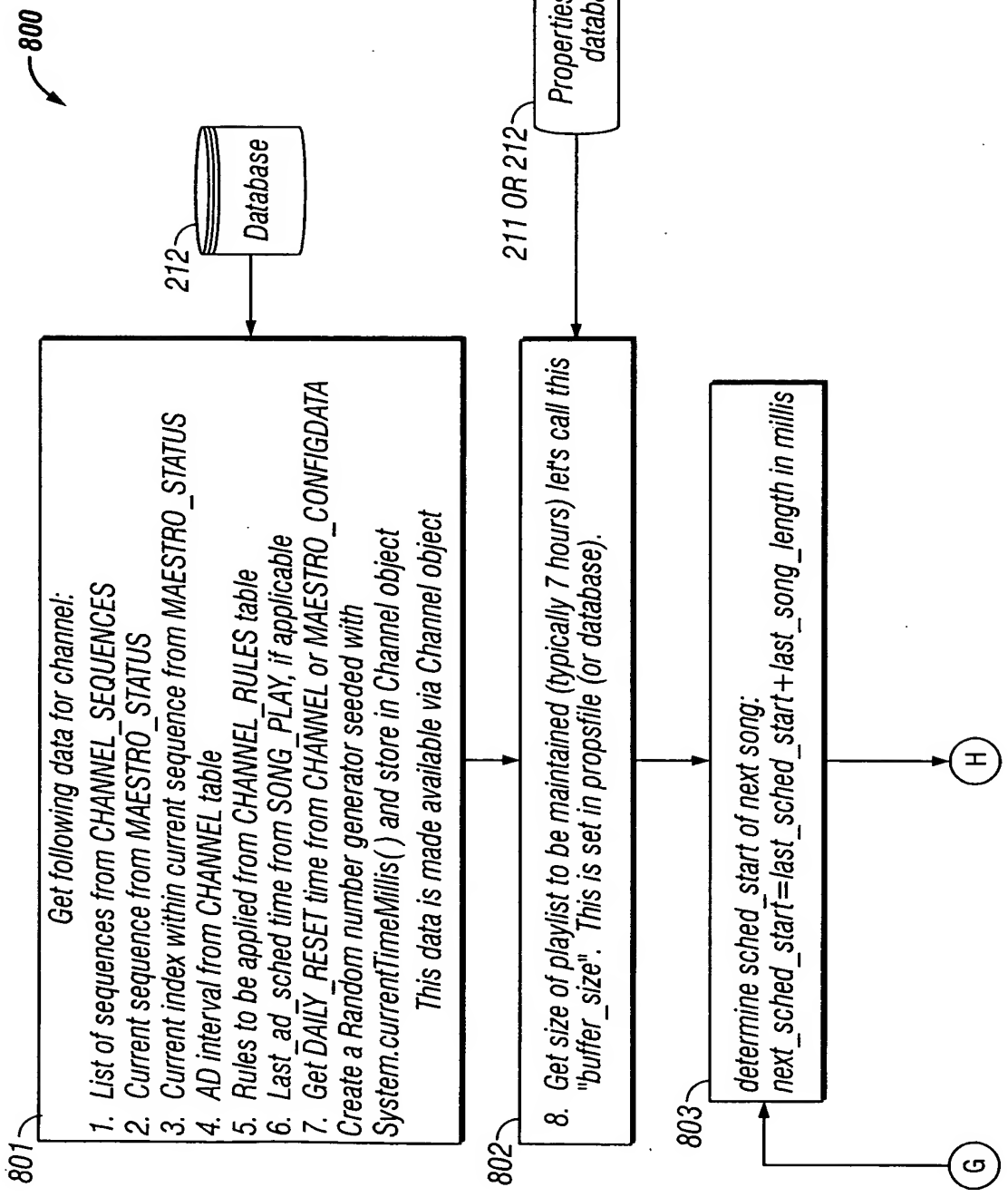

800
Database
212
Properties file or database
211 OR 212
Get following data for channel:
1. List of sequences from CHANNEL_SEQUENCES
2. Current sequence from MAESTRO_STATUS
3. Current index within current sequence from MAESTRO_STATUS
4. AD interval from CHANNEL table
5. Rules to be applied from CHANNEL_RULES table
6. Last_ad_sched time from SONG_PLAY, if applicable
7. Get DAILY_RESET time from CHANNEL or MAESTRO_CONFIGDATA
Create a Random number generator seeded with System.currentTimeMillis( ) and store in Channel object
This data is made available via Channel object
801
8. Get size of playlist to be maintained (typically 7 hours) let's call this "buffer_size". This is set in propsfile (or database).
802
determine sched_start of next song:
next_sched_start=last_sched_start+last_song_length in millis
803
G
H

FIG. 8

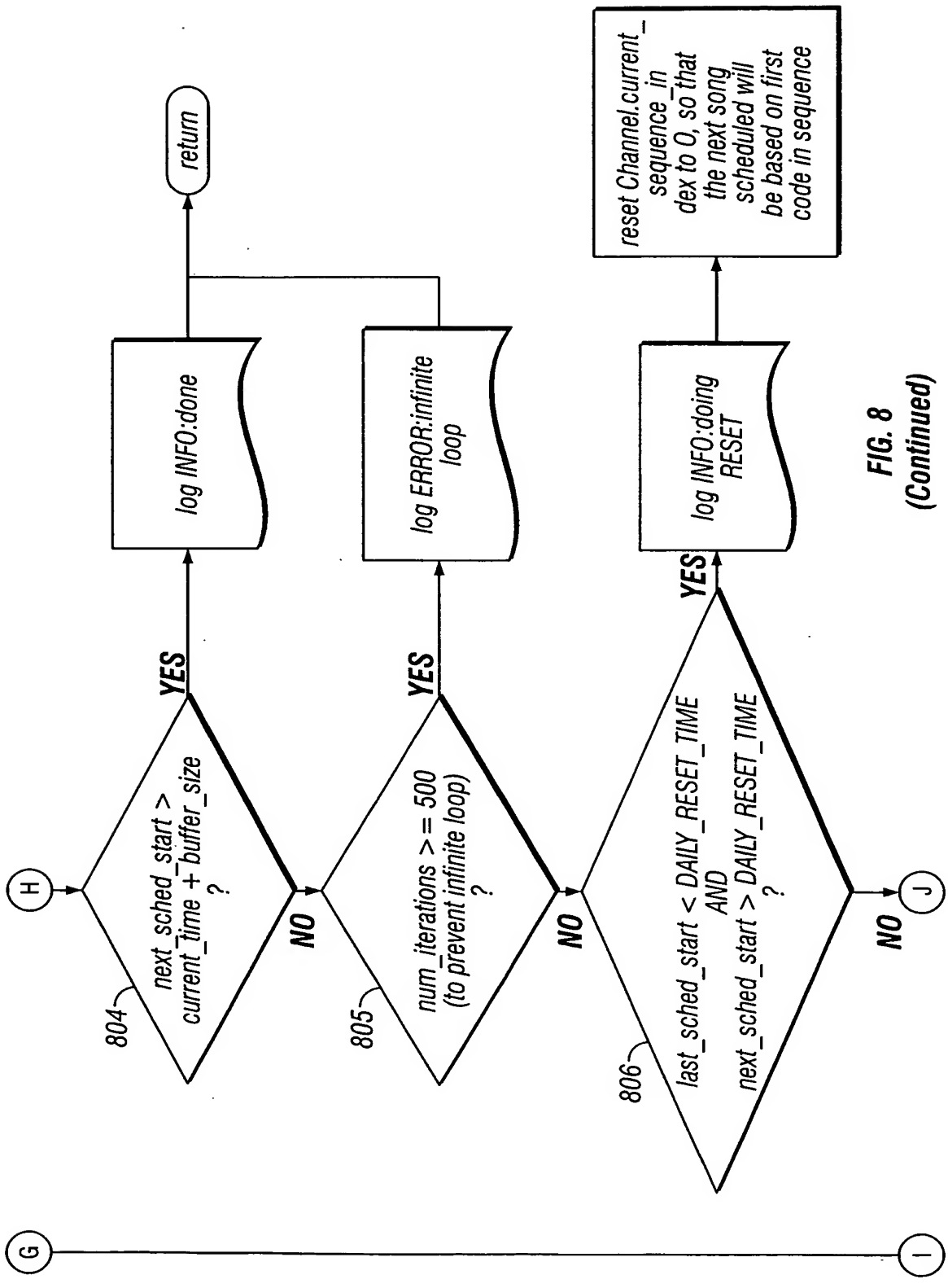


FIG. 8
(Continued)

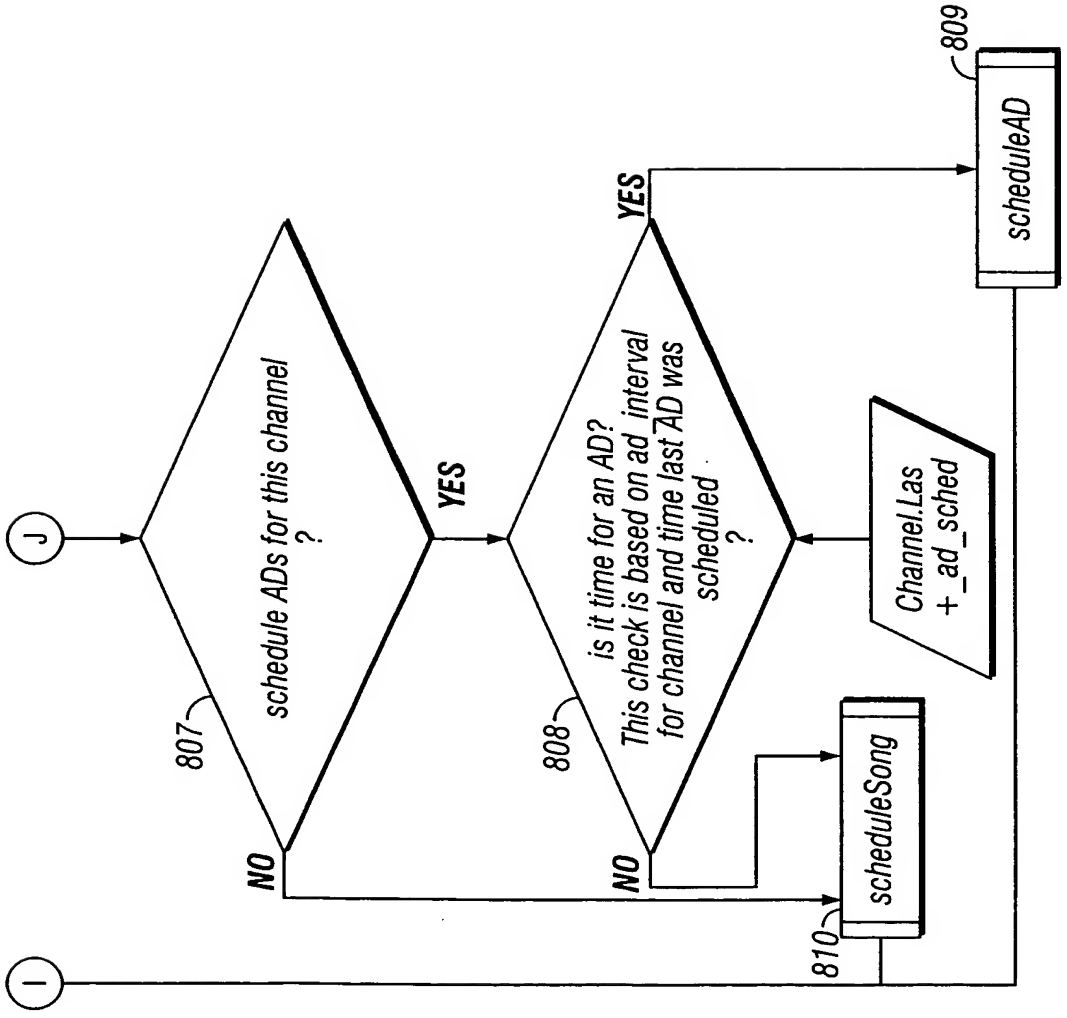


FIG. 8
(Continued)

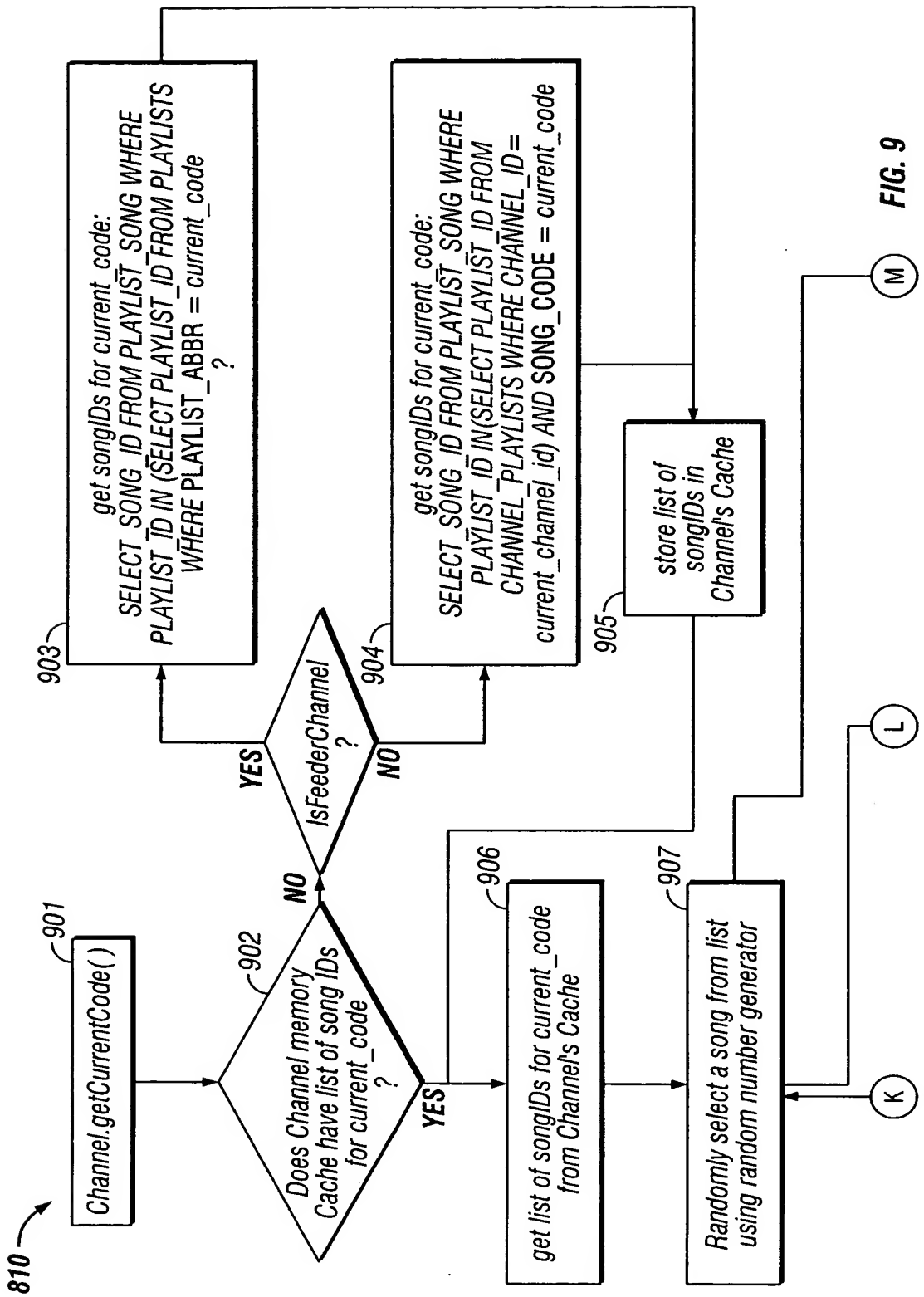

810
Channel.getCurrentCode( )
901
Does Channel memory Cache have list of song IDs for current_code ?
902
NO
YES
IsFeederChannel ?
YES
NO
get songIDs for current_code: SELECT SONG_ID FROM PLAYLIST_SONG WHERE PLAYLIST_ID IN (SELECT PLAYLIST_ID FROM PLAYLISTS WHERE PLAYLIST_ABBR = current_code ?
903
get songIDs for current_code: SELECT SONG_ID FROM PLAYLIST_SONG WHERE PLAYLIST_ID IN(SELECT PLAYLIST_ID FROM CHANNEL_PLAYLISTS WHERE CHANNEL_ID= current_channel_id) AND SONG_CODE = current_code
904
store list of songIDs in Channel's Cache
905
get list of songIDs for current_code from Channel's Cache
906
Randomly select a song from list using random number generator
907
K
L
M

FIG. 9

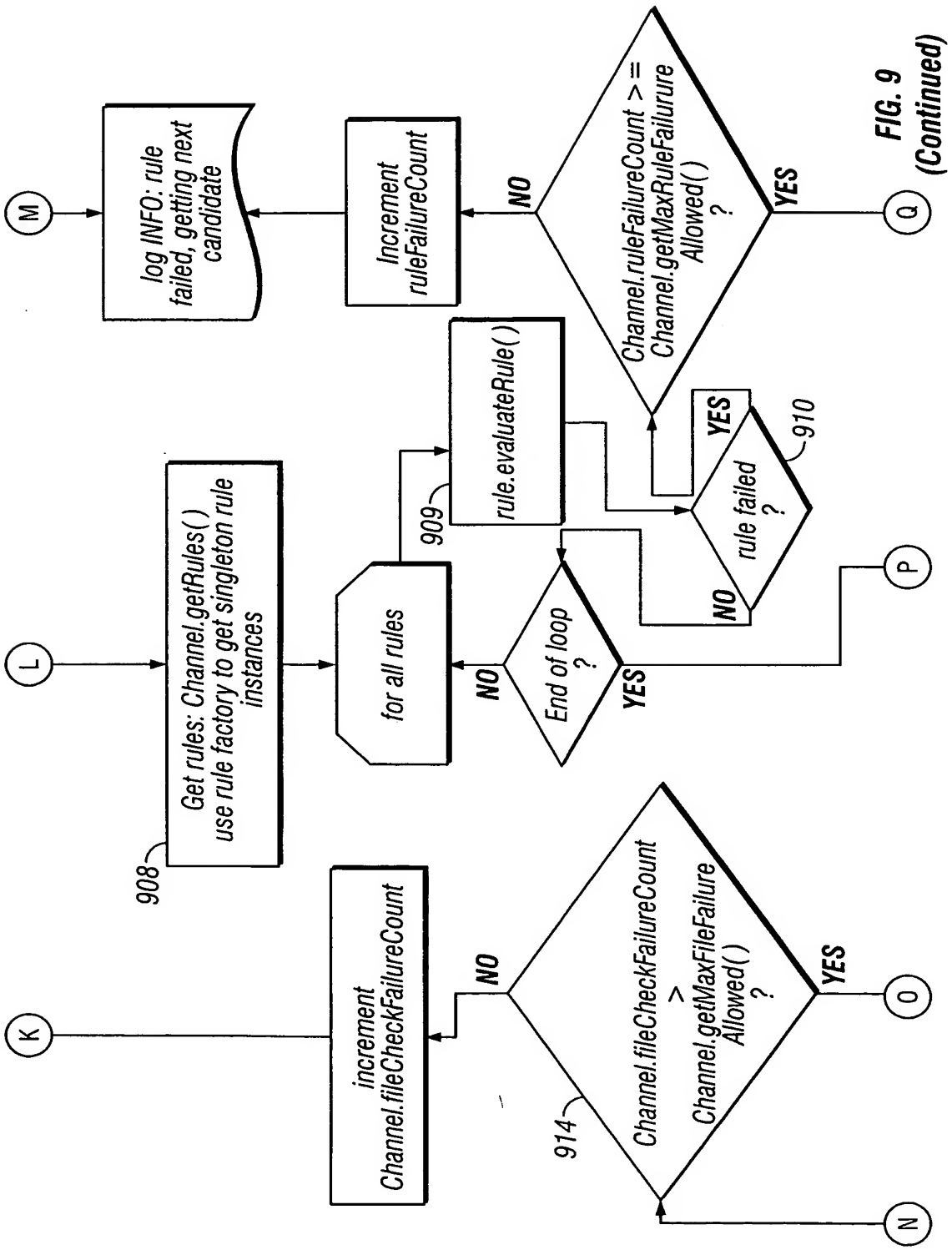


FIG. 9 (Continued)

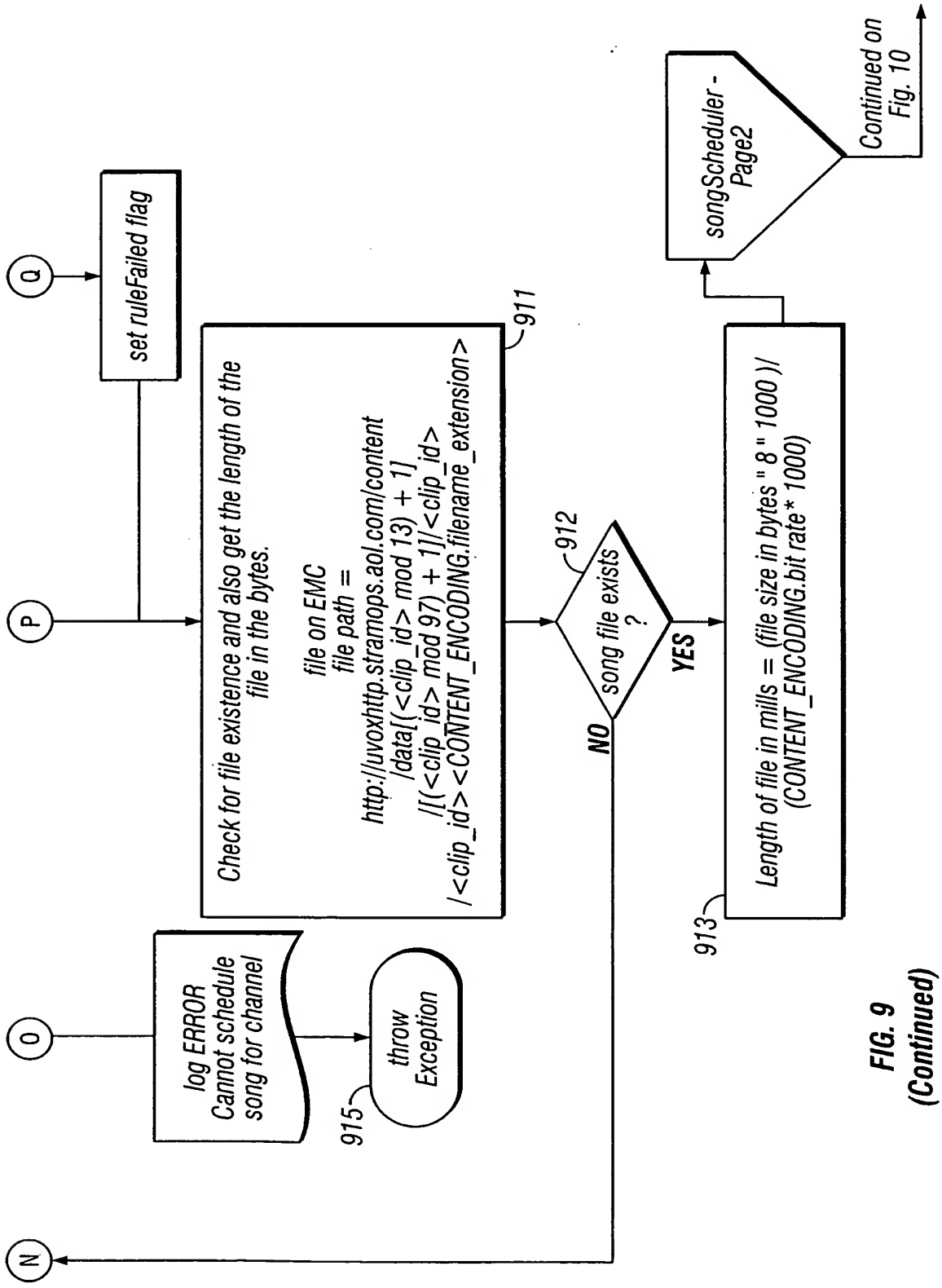


FIG. 9
(Continued)

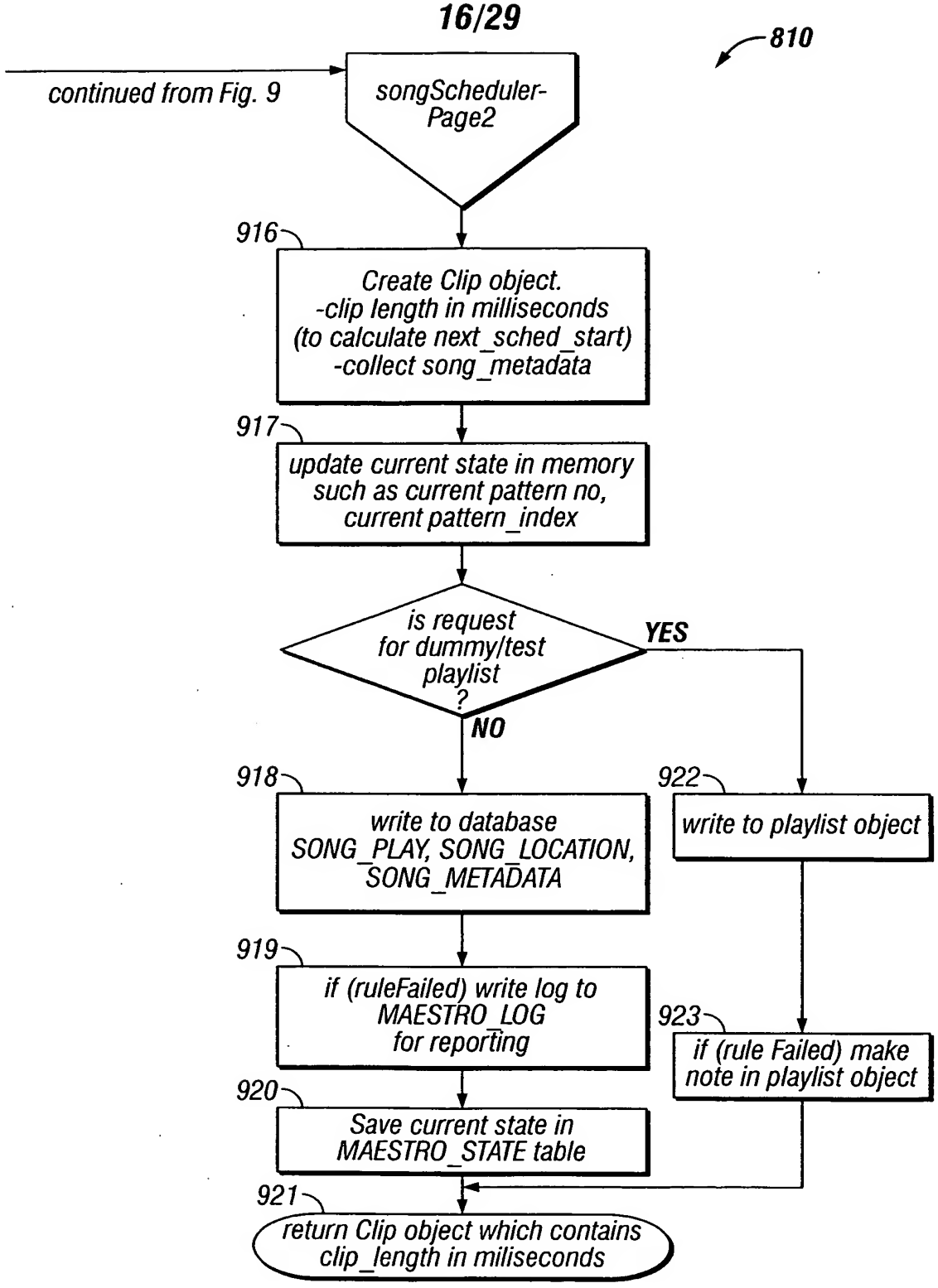

810
continued from Fig. 9
songScheduler-Page2
916
Create Clip object.
-clip length in milliseconds
(to calculate next_sched_start)
-collect song_metadata
917
update current state in memory such as current pattern no, current pattern_index
is request for dummy/test playlist ?
YES
NO
918
write to database SONG_PLAY, SONG_LOCATION, SONG_METADATA
922
write to playlist object
919
if (ruleFailed) write log to MAESTRO_LOG for reporting
923
if (rule Failed) make note in playlist object
920
Save current state in MAESTRO_STATE table
921
return Clip object which contains clip_length in miliseconds

FIG. 10

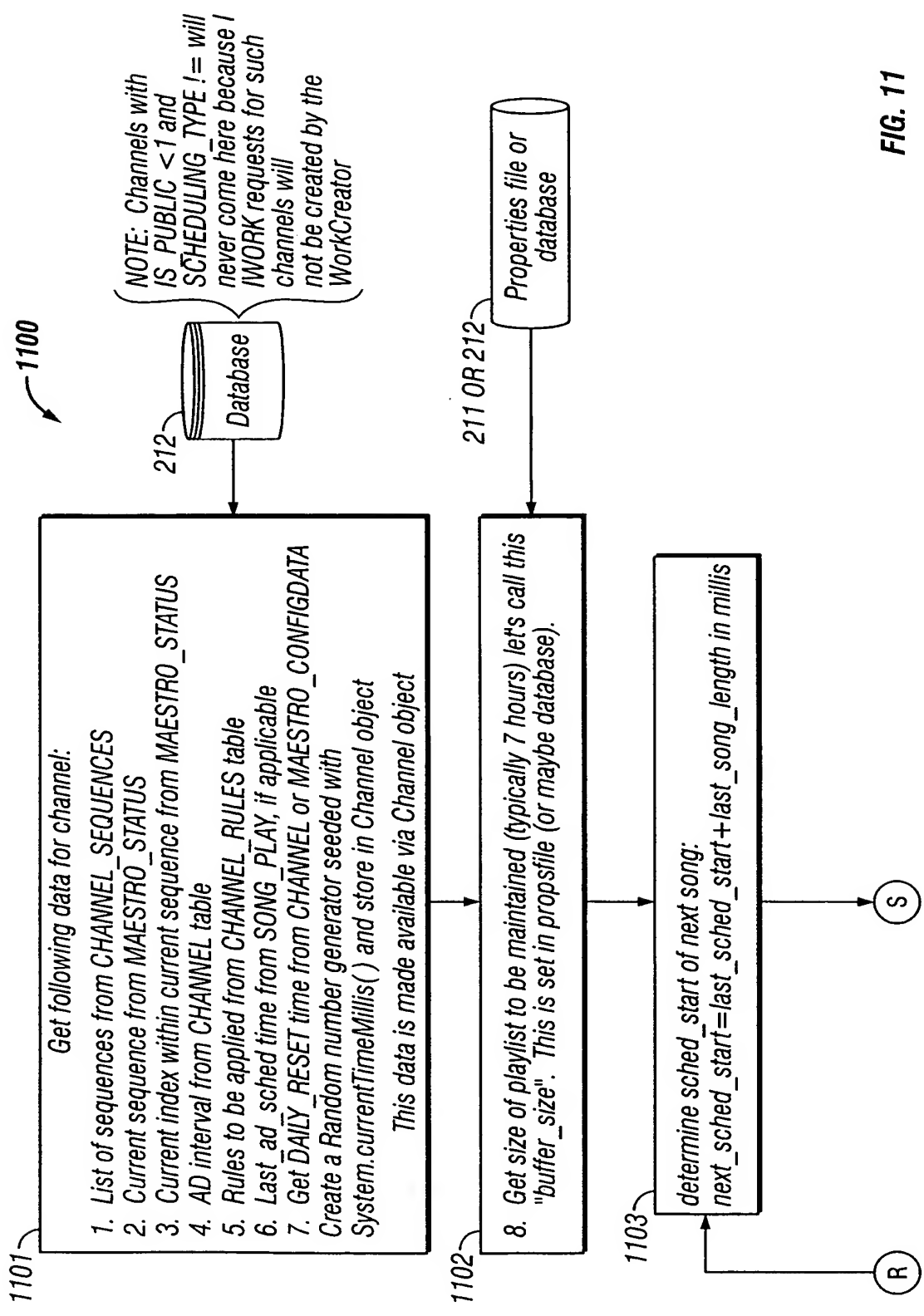

1100
NOTE: Channels with IS_PUBLIC < 1 and SCHEDULING_TYPE != will never come here because IWORK requests for such channels will not be created by the WorkCreator
212
Database
211 OR 212
Properties file or database
1101
Get following data for channel:
1. List of sequences from CHANNEL_SEQUENCES
2. Current sequence from MAESTRO_STATUS
3. Current index within current sequence from MAESTRO_STATUS
4. AD interval from CHANNEL table
5. Rules to be applied from CHANNEL_RULES table
6. Last_ad_sched time from SONG_PLAY, if applicable
7. Get DAILY_RESET time from CHANNEL or MAESTRO_CONFIGDATA
Create a Random number generator seeded with System.currentTimeMillis( ) and store in Channel object
This data is made available via Channel object
1102
8. Get size of playlist to be maintained (typically 7 hours) let's call this "buffer_size". This is set in propsfile (or maybe database).
1103
determine sched_start of next song:
next_sched_start=last_sched_start+last_song_length in millis
R
S

FIG. 11

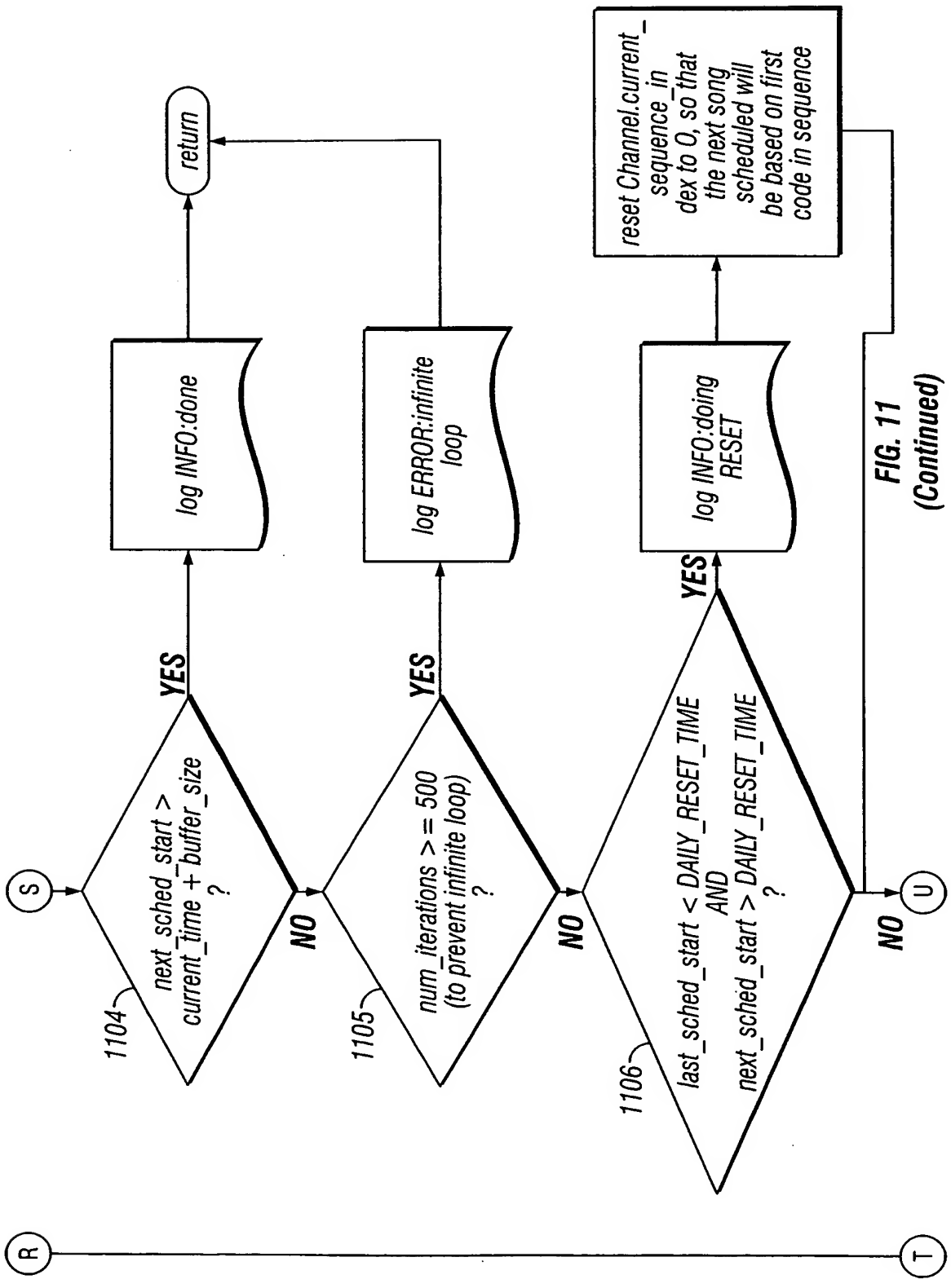
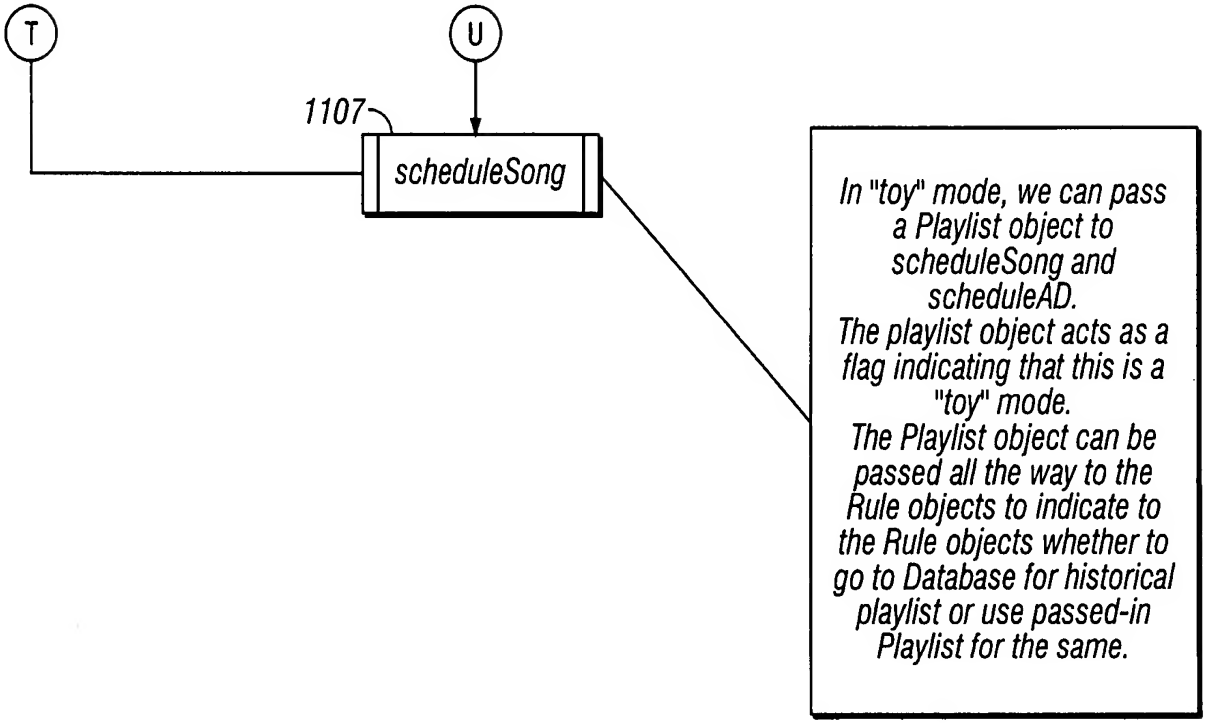


FIG. 11
(Continued)



Note on AD Scheduling:
for version 1.0 of Maestro , AD scheduling will be done by a separate thread in the system, not the SongScheduler.
When we integrate with LightingCast, AD scheduling will be done as part of the SongScheduling logic.

LEGEND:
ADs will be scheduled using CDPIA for clipbased channels in v1.0 of Maestro

FIG. 11
(Continued)

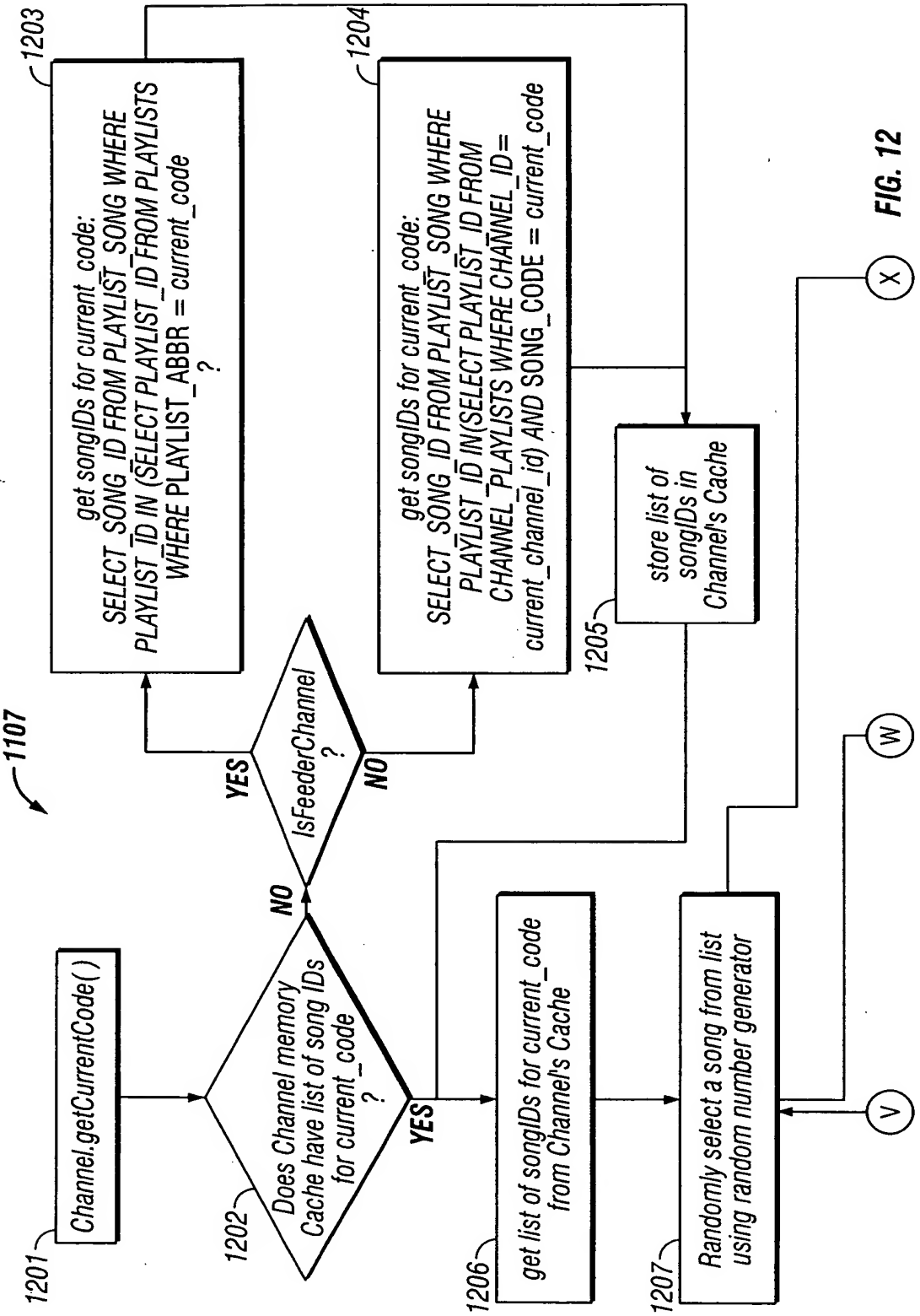

1107
1201
Channel.getCurrentCode( )
1202
Does Channel memory Cache have list of song IDs for current_code ?
NO
YES
IsFeederChannel ?
YES
NO
1203
get songIDs for current_code: SELECT_SONG_ID FROM PLAYLIST_SONG WHERE PLAYLIST_ID IN (SELECT PLAYLIST_ID FROM PLAYLISTS WHERE PLAYLIST_ABBR = current_code ?
1204
get songIDs for current_code: SELECT_SONG_ID FROM PLAYLIST_SONG WHERE PLAYLIST_ID IN(SELECT PLAYLIST_ID FROM CHANNEL_PLAYLISTS WHERE CHANNEL_ID= current_channel_id) AND SONG_CODE = current_code
1205
store list of songIDs in Channel's Cache
1206
get list of songIDs for current_code from Channel's Cache
1207
Randomly select a song from list using random number generator
V
W
X

FIG. 12

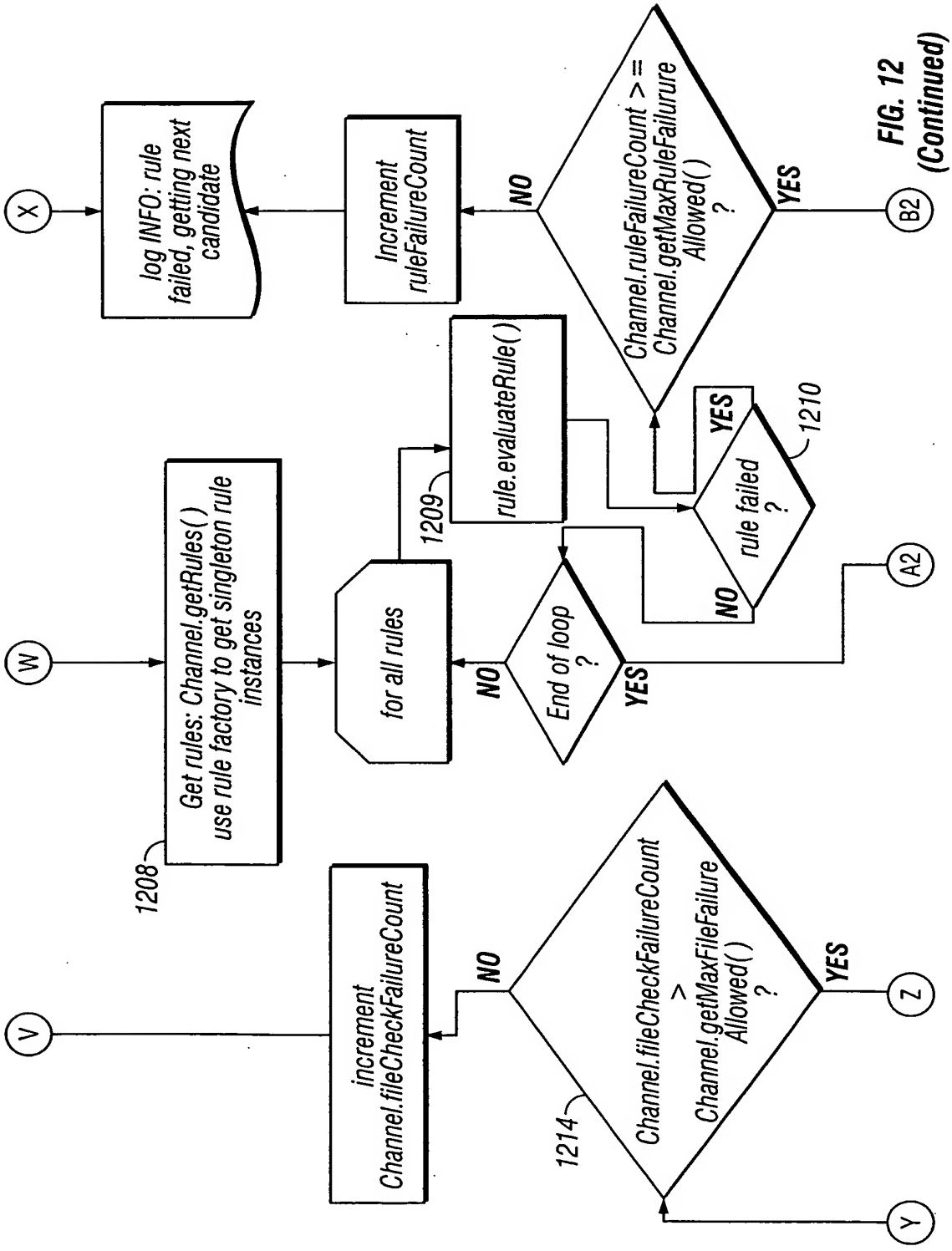


FIG. 12 (Continued)

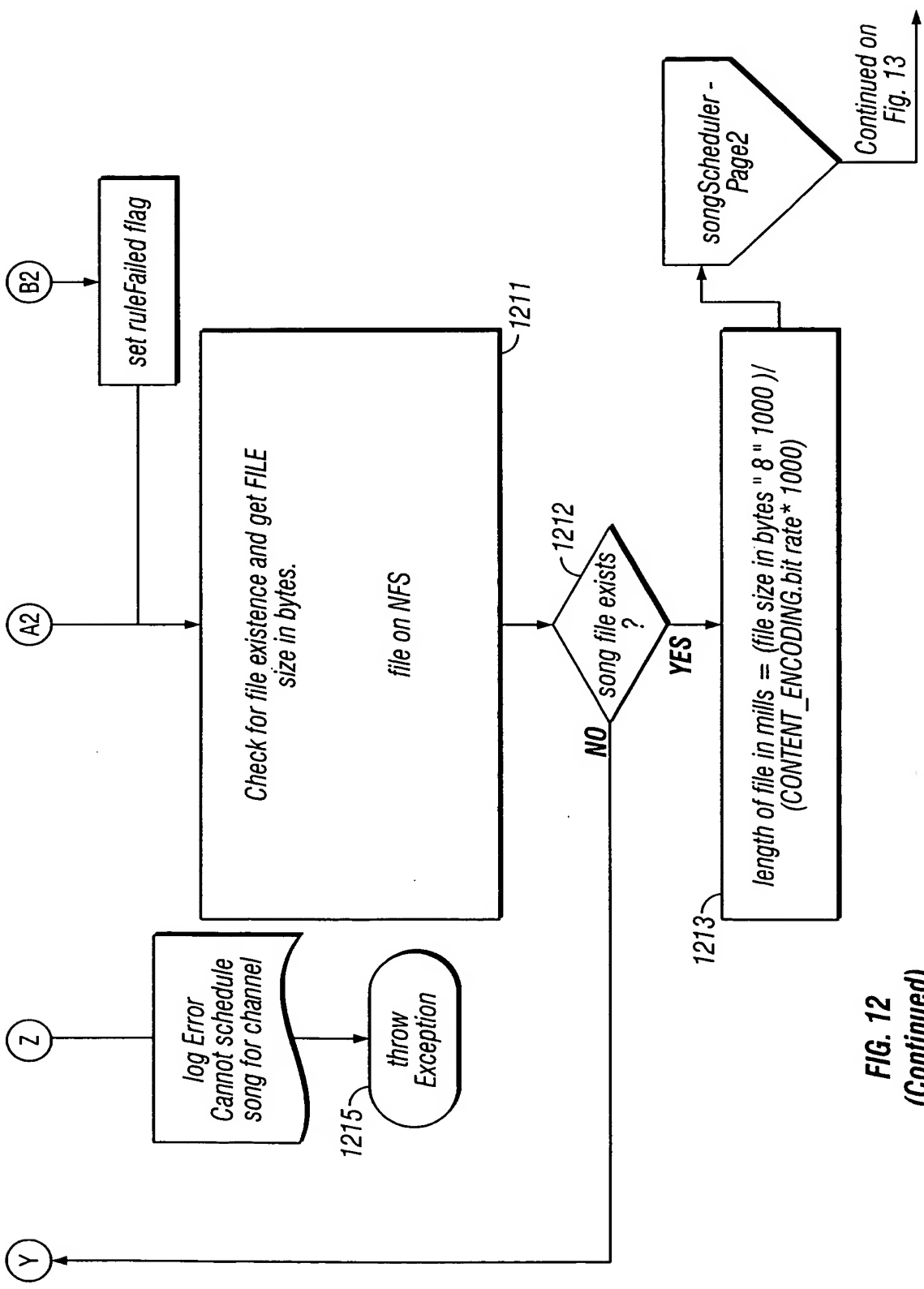


FIG. 12
(Continued)

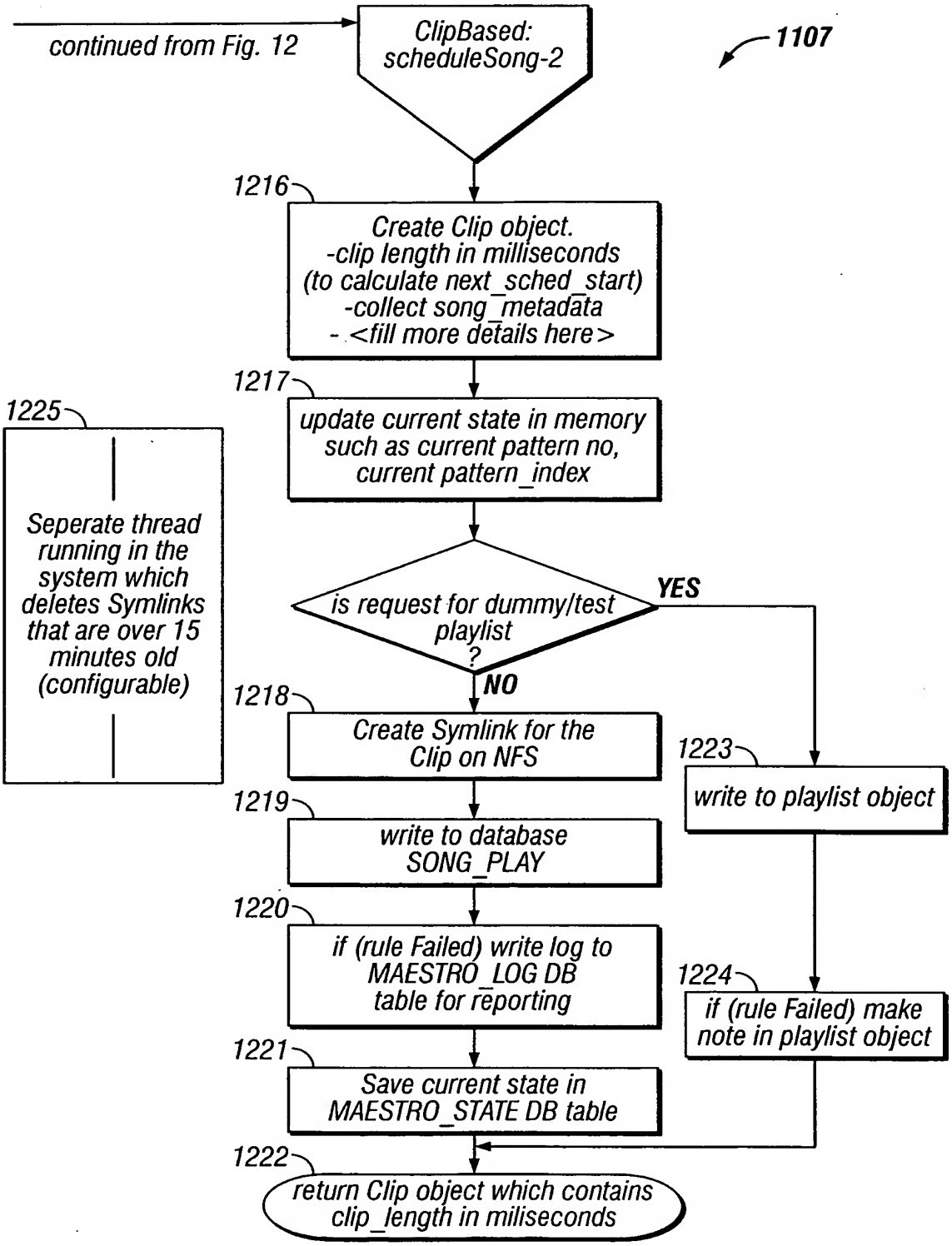

continued from Fig. 12
ClipBased: scheduleSong-2
1107
1216
Create Clip object.
-clip length in milliseconds (to calculate next_sched_start)
-collect song_metadata
- <fill more details here>
1217
update current state in memory such as current pattern no, current pattern_index
1225
Seperate thread running in the system which deletes Symlinks that are over 15 minutes old (configurable)
is request for dummy/test playlist ?
YES
NO
1218
Create Symlink for the Clip on NFS
1223
write to playlist object
1219
write to database SONG_PLAY
1220
if (rule Failed) write log to MAESTRO_LOG DB table for reporting
1224
if (rule Failed) make note in playlist object
1221
Save current state in MAESTRO_STATE DB table
1222
return Clip object which contains clip_length in miliseconds

FIG. 13

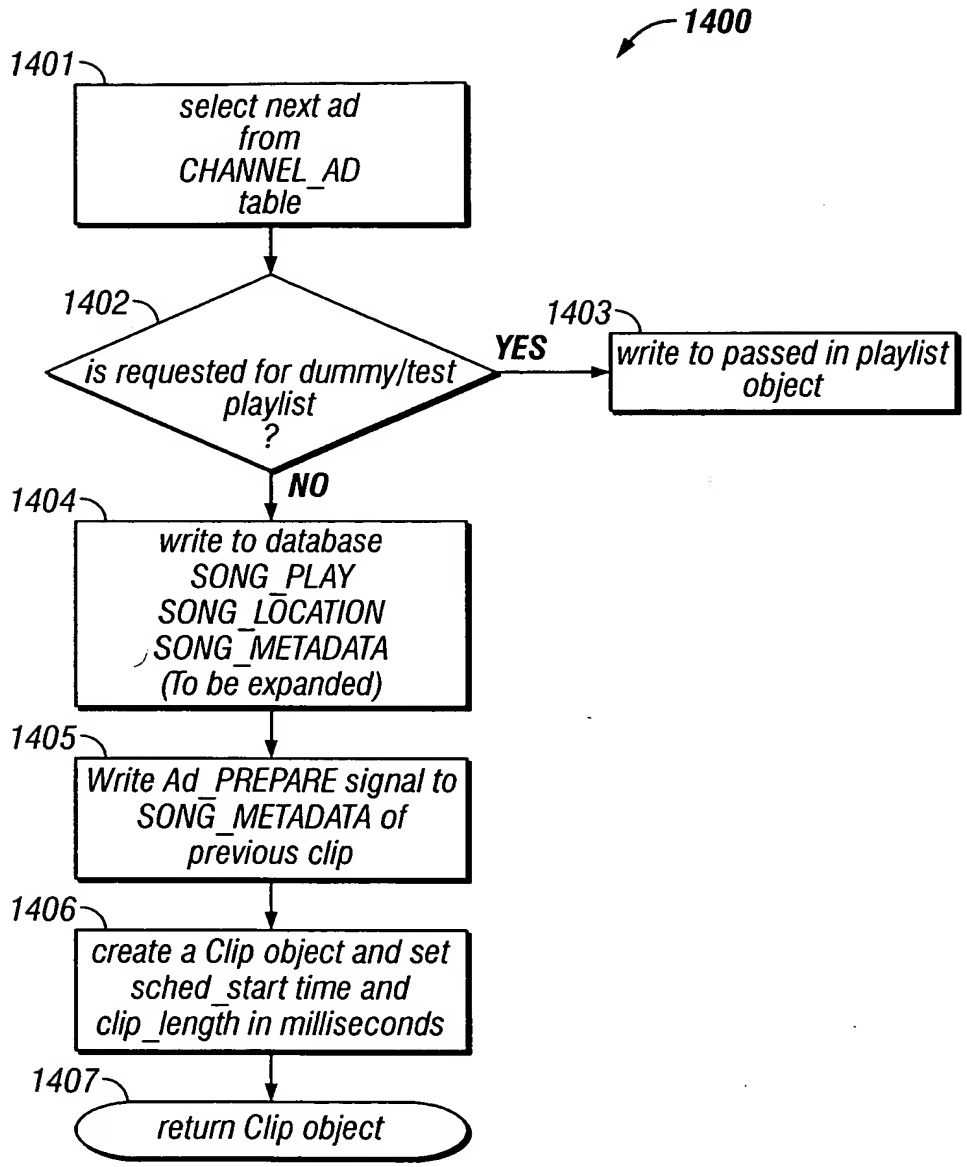

1400
1401
select next ad from CHANNEL_AD table
1402
is requested for dummy/test playlist ?
YES
1403
write to passed in playlist object
NO
1404
write to database SONG_PLAY SONG_LOCATION SONG_METADATA (To be expanded)
1405
Write Ad_PREPARE signal to SONG_METADATA of previous clip
1406
create a Clip object and set sched_start time and clip_length in milliseconds
1407
return Clip object

FIG. 14

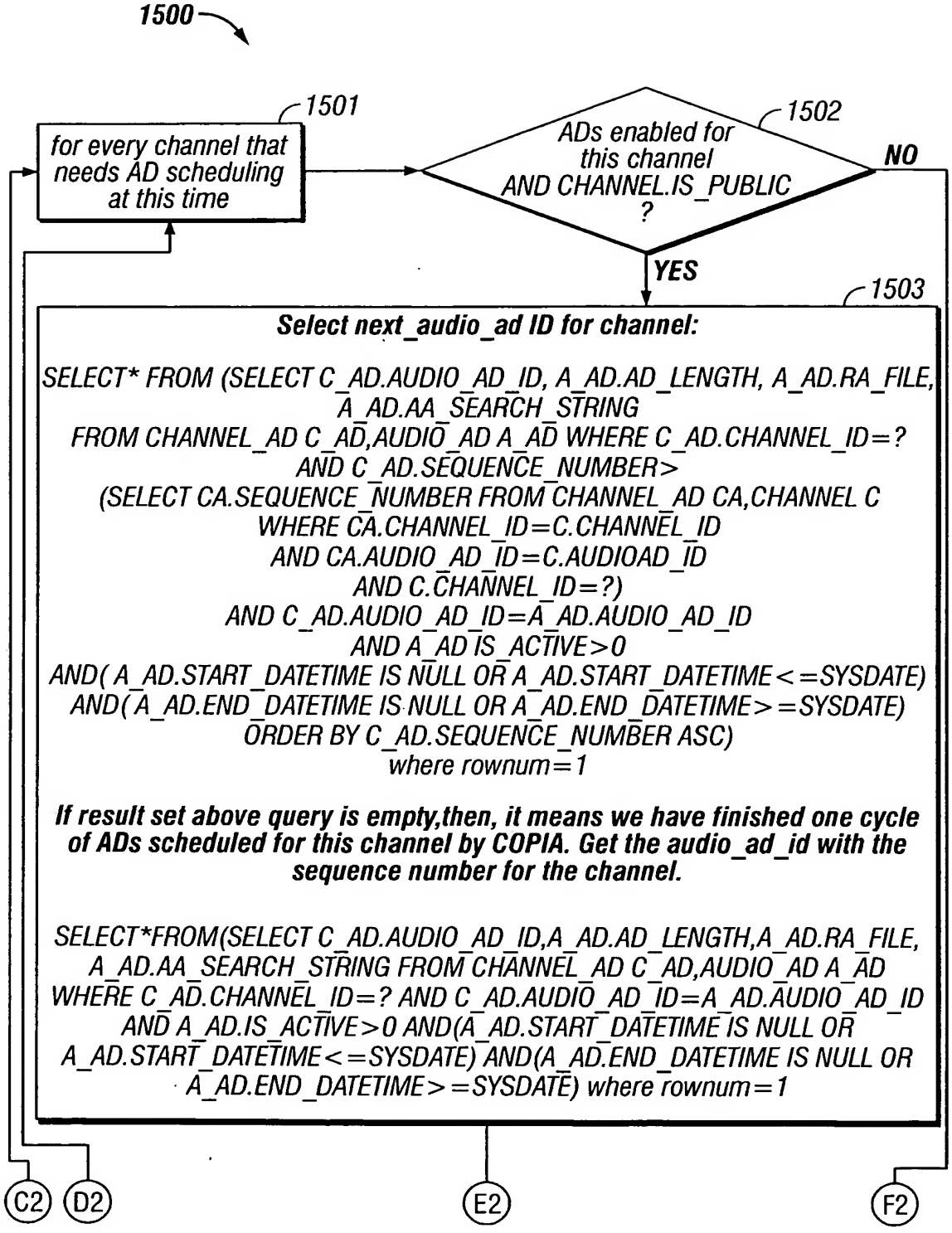

1500
1501
for every channel that needs AD scheduling at this time
1502
ADs enabled for this channel AND CHANNEL.IS_PUBLIC ?
NO
YES
1503
Select next_audio_ad ID for channel:
SELECT* FROM (SELECT C_AD.AUDIO_AD_ID, A_AD.AD_LENGTH, A_AD.RA_FILE, A_AD.AA_SEARCH_STRING
FROM CHANNEL_AD C_AD,AUDIO_AD A_AD WHERE C_AD.CHANNEL_ID=?
AND C_AD.SEQUENCE_NUMBER>
(SELECT CA.SEQUENCE_NUMBER FROM CHANNEL_AD CA,CHANNEL C
WHERE CA.CHANNEL_ID=C.CHANNEL_ID
AND CA.AUDIO_AD_ID=C.AUDIOAD_ID
AND C.CHANNEL_ID=?)
AND C_AD.AUDIO_AD_ID=A_AD.AUDIO_AD_ID
AND A_AD.IS_ACTIVE>0
AND(A_AD.START_DATETIME IS NULL OR A_AD.START_DATETIME<=SYSDATE)
AND(A_AD.END_DATETIME IS NULL OR A_AD.END_DATETIME>=SYSDATE)
ORDER BY C_AD.SEQUENCE_NUMBER ASC)
where rownum=1
If result set above query is empty,then, it means we have finished one cycle of ADs scheduled for this channel by COPIA. Get the audio_ad_id with the sequence number for the channel.
SELECT*FROM(SELECT C_AD.AUDIO_AD_ID,A_AD.AD_LENGTH,A_AD.RA_FILE, A_AD.AA_SEARCH_STRING FROM CHANNEL_AD C_AD,AUDIO_AD A_AD
WHERE C_AD.CHANNEL_ID=? AND C_AD.AUDIO_AD_ID=A_AD.AUDIO_AD_ID
AND A_AD.IS_ACTIVE>0 AND(A_AD.START_DATETIME IS NULL OR
A_AD.START_DATETIME<=SYSDATE) AND(A_AD.END_DATETIME IS NULL OR
A_AD.END_DATETIME>=SYSDATE) where rownum=1
C2
D2
E2
F2

FIG. 15

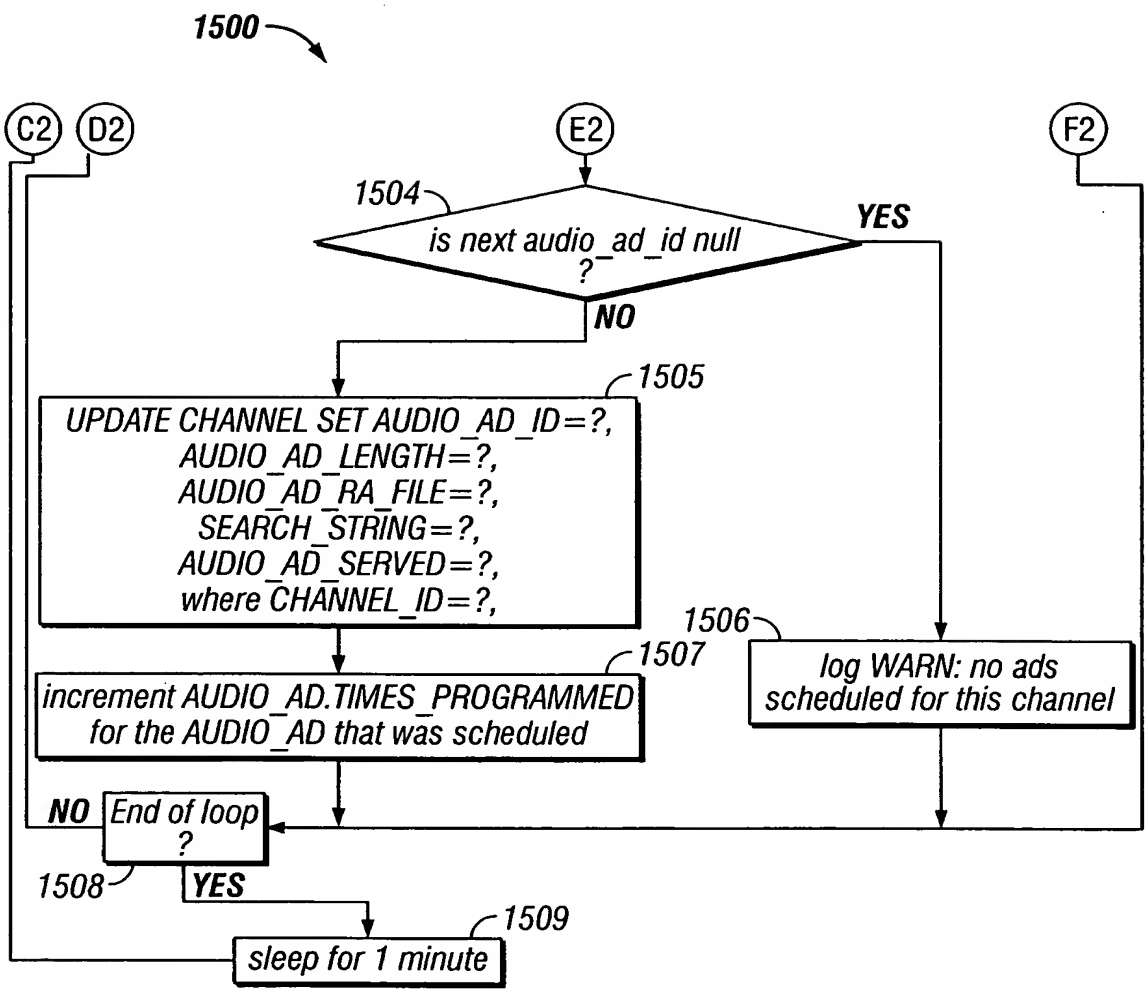


FIG. 15
(Continued)

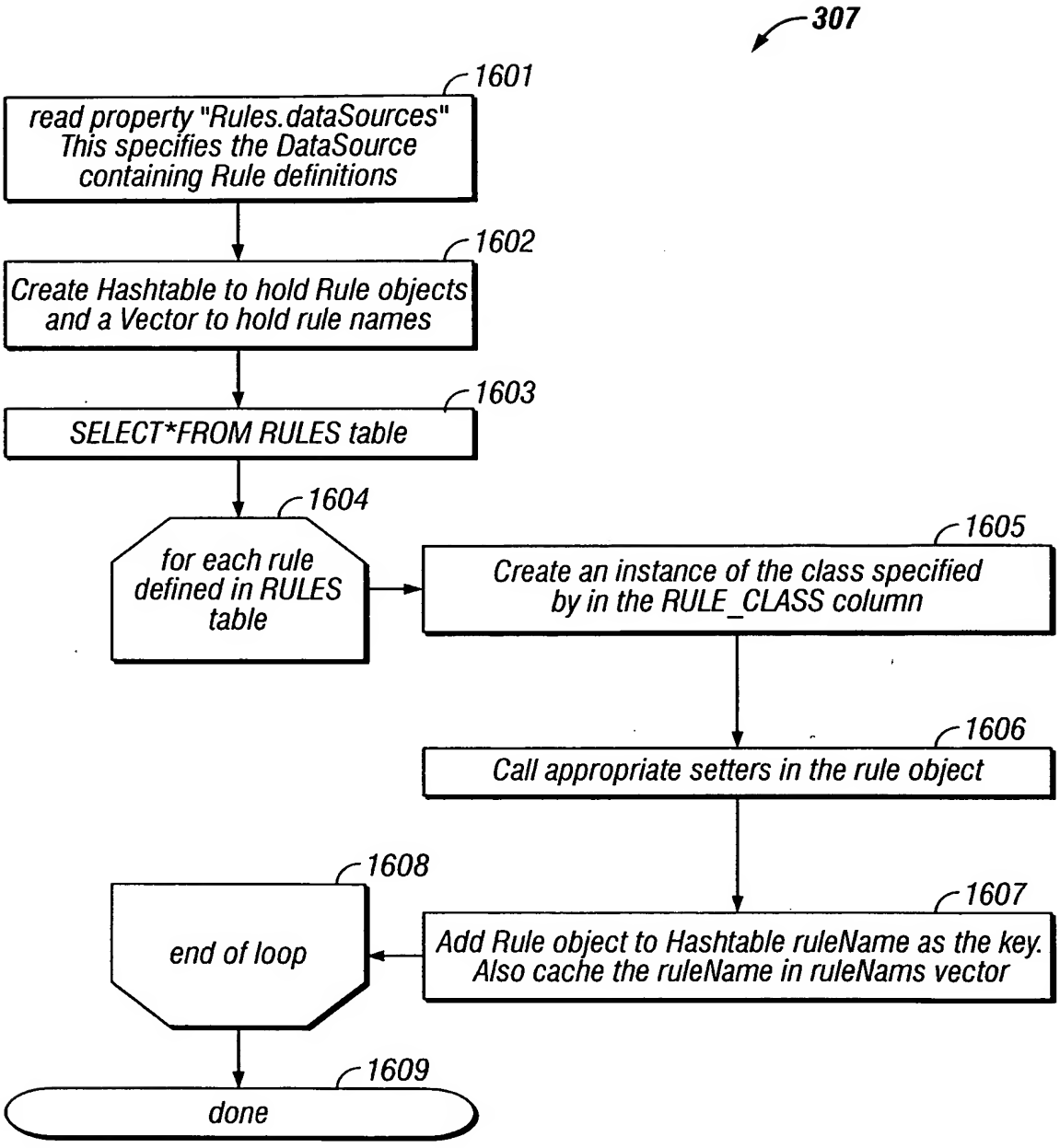

307
1601
read property "Rules.dataSources" This specifies the DataSource containing Rule definitions
1602
Create Hashtable to hold Rule objects and a Vector to hold rule names
1603
SELECT*FROM RULES table
1604
for each rule defined in RULES table
1605
Create an instance of the class specified by in the RULE_CLASS column
1606
Call appropriate setters in the rule object
1607
Add Rule object to Hashtable ruleName as the key. Also cache the ruleName in ruleNams vector
1608
end of loop
1609
done

FIG. 16

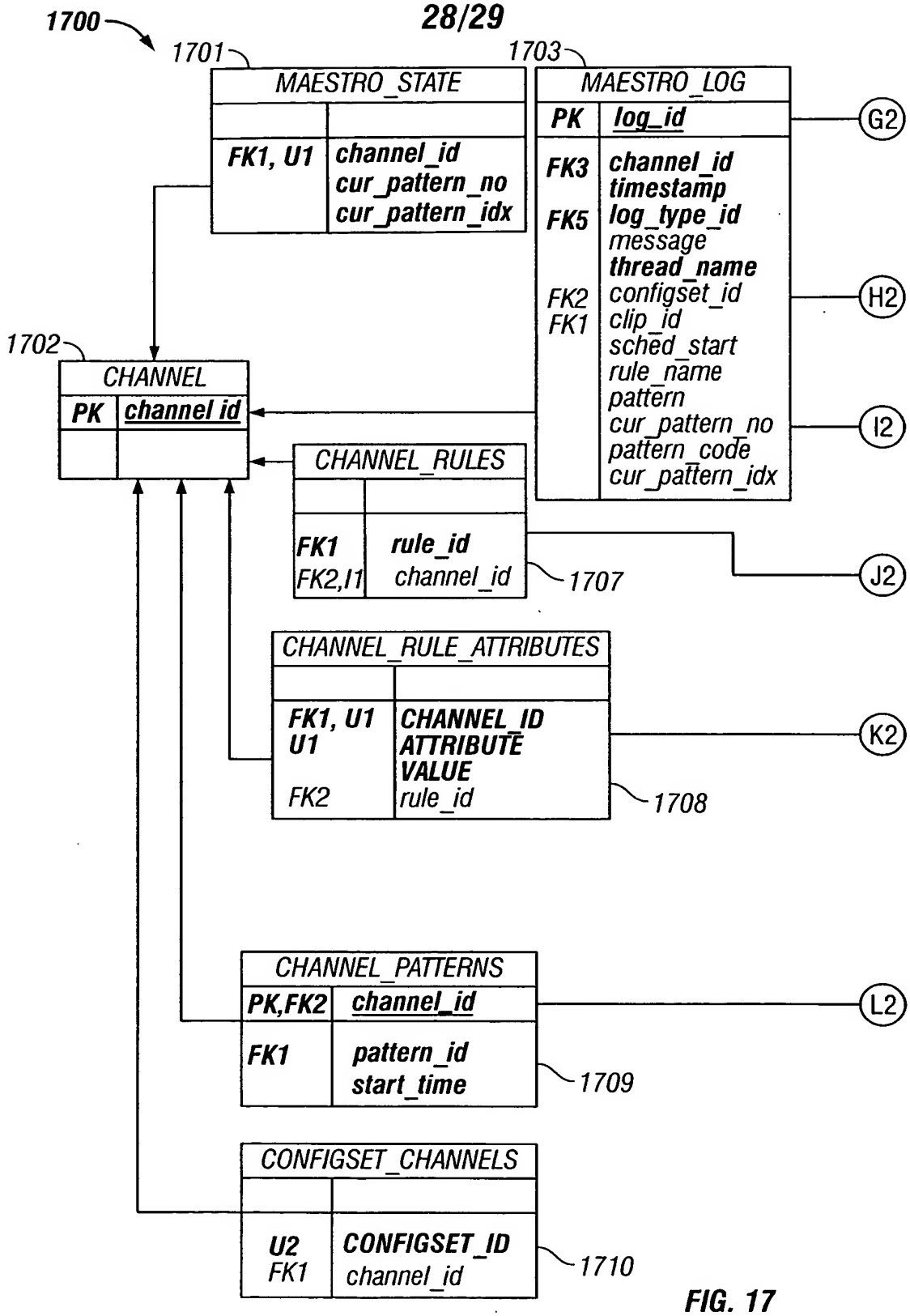

1700
1701
MAESTRO_STATE
FK1, U1
channel_id
cur_pattern_no
cur_pattern_idx
1703
MAESTRO_LOG
PK
log_id
FK3
channel_id
timestamp
FK5
log_type_id
message
thread_name
FK2
configset_id
FK1
clip_id
sched_start
rule_name
pattern
cur_pattern_no
pattern_code
cur_pattern_idx
G2
H2
I2
1702
CHANNEL
PK
channel id
CHANNEL_RULES
FK1
rule_id
FK2,I1
channel_id
1707
J2
CHANNEL_RULE_ATTRIBUTES
FK1, U1
CHANNEL_ID
U1
ATTRIBUTE
VALUE
FK2
rule_id
1708
K2
CHANNEL_PATTERNS
PK,FK2
channel_id
FK1
pattern_id
start_time
1709
L2
CONFIGSET_CHANNELS
U2
CONFIGSET_ID
FK1
channel_id
1710

FIG. 17

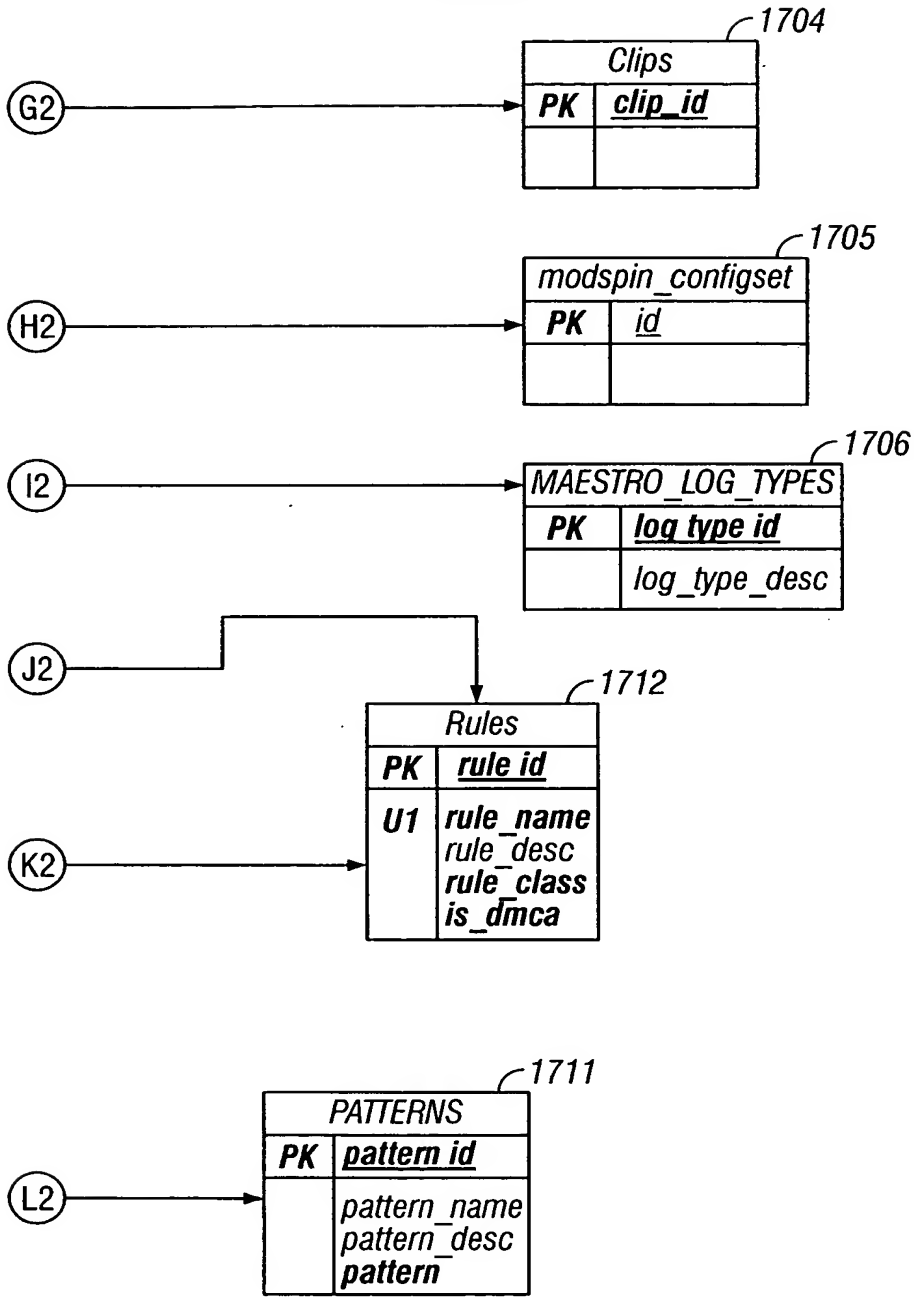


FIG. 17
(Continued)